

DATA 221 Homework 2

Chris Low

Problem 1

1(a)

Calculate

$$P(\text{WB score} = \text{Improved}, \text{PCC symptoms} = \text{Less} \mid \text{Vaccinated})$$

```
import pandas as pd

train = pd.read_csv("data/PCC_study_train.csv")

vacc = train[train["vax_status"] == "Vaccinated"]

p_vacc = (
    (vacc["WBscore"] == "Improved") &
    (vacc["PCCsymp"] == "Less")
).mean()

print(p_vacc)
```

0.75

1(b)

Calculate

$$P(\text{WB score} = \text{Improved}, \text{PCC symptoms} = \text{Less} \mid \text{Unvaccinated})$$

```

unvacc = train[train["vax_status"] == "Unvaccinated"]

p_unvacc = (
    (unvacc["WBscore"] == "Improved") &
    (unvacc["PCCsymp"] == "Less")
).mean()

print(p_unvacc)

```

0.07692307692307693

1(c)

```

import pandas as pd

def make_table(df, label):
    wb = (
        df["WBscore"]
        .value_counts(normalize=True)
        .rename("Probability")
        .reset_index()
        .rename(columns={"index": "WBscore"})
    )

    pcc = (
        df["PCCsymp"]
        .value_counts(normalize=True)
        .rename("Probability")
        .reset_index()
        .rename(columns={"index": "PCCsymp"})
    )

    print(f"\n{n[label]} - Well-being score")
    display(wb)

    print(f"\n{n[label]} - PCC symptoms")
    display(pcc)

```

```
make_table(vacc, "Vaccinated")
make_table(unvacc, "Unvaccinated")
```

Vaccinated - Well-being score

	WBscore	Probability
0	Improved	0.777778
1	Unchanged	0.138889
2	Worsened	0.083333

Vaccinated - PCC symptoms

	PCCsymp	Probability
0	Less	0.861111
1	Same	0.083333
2	More	0.055556

Unvaccinated - Well-being score

	WBscore	Probability
0	Unchanged	0.512821
1	Worsened	0.307692
2	Improved	0.179487

Unvaccinated - PCC symptoms

	PCCsymp	Probability
0	Same	0.512821
1	More	0.282051

	PCCsymp	Probability
2	Less	0.205128

1(d)

Assuming conditional independence, write a function that computes

$$P(\text{outcome} \mid \text{class})$$

```
def naive_bayes_likelihood(
    wb_score,
    pcc_symp,
    wb_probs,
    pcc_probs,
    class_prob
):
    """
    wb_score: The observed WBScore (e.g., "Improved")
    pcc_symp: The observed PCCsymp (e.g., "Less")
    wb_probs: Dictionary of conditional probs for WBScore given the class
    pcc_probs: Dictionary of conditional probs for PCCsymp given the class
    class_prob: The prior probability of the class (e.g., P(Vaccinated))
    """
    return (
        wb_probs[wb_score] *
        pcc_probs[pcc_symp] *
        class_prob
    )
```

1(e)

```
p_vaccinated = (train["vax_status"] == "Vaccinated").mean()
p_unvaccinated = (train["vax_status"] == "Unvaccinated").mean()

vacc_wb_probs = vacc["WBScore"].value_counts(normalize=True)
vacc_pcc_probs = vacc["PCCsymp"].value_counts(normalize=True)

unvacc_wb_probs = unvacc["WBScore"].value_counts(normalize=True)
unvacc_pcc_probs = unvacc["PCCsymp"].value_counts(normalize=True)
```

```

# likelihoods for WB = Worsened, PCC = Same
lik_vacc = naive_bayes_likelihood(
    "Worsened",
    "Same",
    vacc_wb_probs,
    vacc_pcc_probs,
    p_vaccinated
)

lik_unvacc = naive_bayes_likelihood(
    "Worsened",
    "Same",
    unvacc_wb_probs,
    unvacc_pcc_probs,
    p_unvaccinated
)

print("Vaccinated likelihood:", lik_vacc)
print("Unvaccinated likelihood:", lik_unvacc)

```

Vaccinated likelihood: 0.003333333333333333
 Unvaccinated likelihood: 0.08205128205128205

1(f)

```

test = pd.read_csv("data/PCC_study_test.csv")

def predict_vax_status(row):
    lv = naive_bayes_likelihood(
        row["WBscore"],
        row["PCCsymp"],
        vacc_wb_probs,
        vacc_pcc_probs,
        p_vaccinated
    )
    lu = naive_bayes_likelihood(
        row["WBscore"],
        row["PCCsymp"],
        unvacc_wb_probs,

```

```

        unvacc_pcc_probs,
        p_unvaccinated
    )
    return "Vaccinated" if lv > lu else "Unvaccinated"

test["predicted_vax_status"] = test.apply(predict_vax_status, axis=1)

test.head()

```

	Unnamed: 0	vax_status2	WBscore	PCCsymp	predicted_vax_status
0	1	Vaccinated	Improved	Less	Vaccinated
1	2	Vaccinated	Improved	Less	Vaccinated
2	3	Vaccinated	Improved	Less	Vaccinated
3	4	Vaccinated	Worsened	Less	Vaccinated
4	5	Vaccinated	Unchanged	Less	Vaccinated

1(g)

```

from sklearn.naive_bayes import CategoricalNB
from sklearn.preprocessing import LabelEncoder

label_wb = LabelEncoder()
label_pcc = LabelEncoder()
label_y = LabelEncoder()

X_train = pd.DataFrame({
    "WBscore": label_wb.fit_transform(train["WBscore"]),
    "PCCsymp": label_pcc.fit_transform(train["PCCsymp"])
})

y_train = label_y.fit_transform(train["vax_status"])

X_test = pd.DataFrame({
    "WBscore": label_wb.transform(test["WBscore"]),
    "PCCsymp": label_pcc.transform(test["PCCsymp"])
})

model = CategoricalNB()
model.fit(X_train, y_train)

```

```

sklearn_preds = label_y.inverse_transform(model.predict(X_test))

# KEY: Compare with manual predictions
comparison = test["predicted_vax_status"].values == sklearn_preds

print("Are all predictions identical?", comparison.all())

```

Are all predictions identical? True

We see that running this code confirms a 100% match between the manual calculation and the Scikit-Learn implementation.

Problem 2

```

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from scipy.stats import norm

```

2(a)

```

# Load Data
iris = pd.read_csv('data/iris.csv')
df = iris.copy()

# Split (80% Train, 20% Test)
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)

print(f"Training set size: {len(train_df)}")
print(f"Test set size: {len(test_df)}")

```

Training set size: 120
Test set size: 30

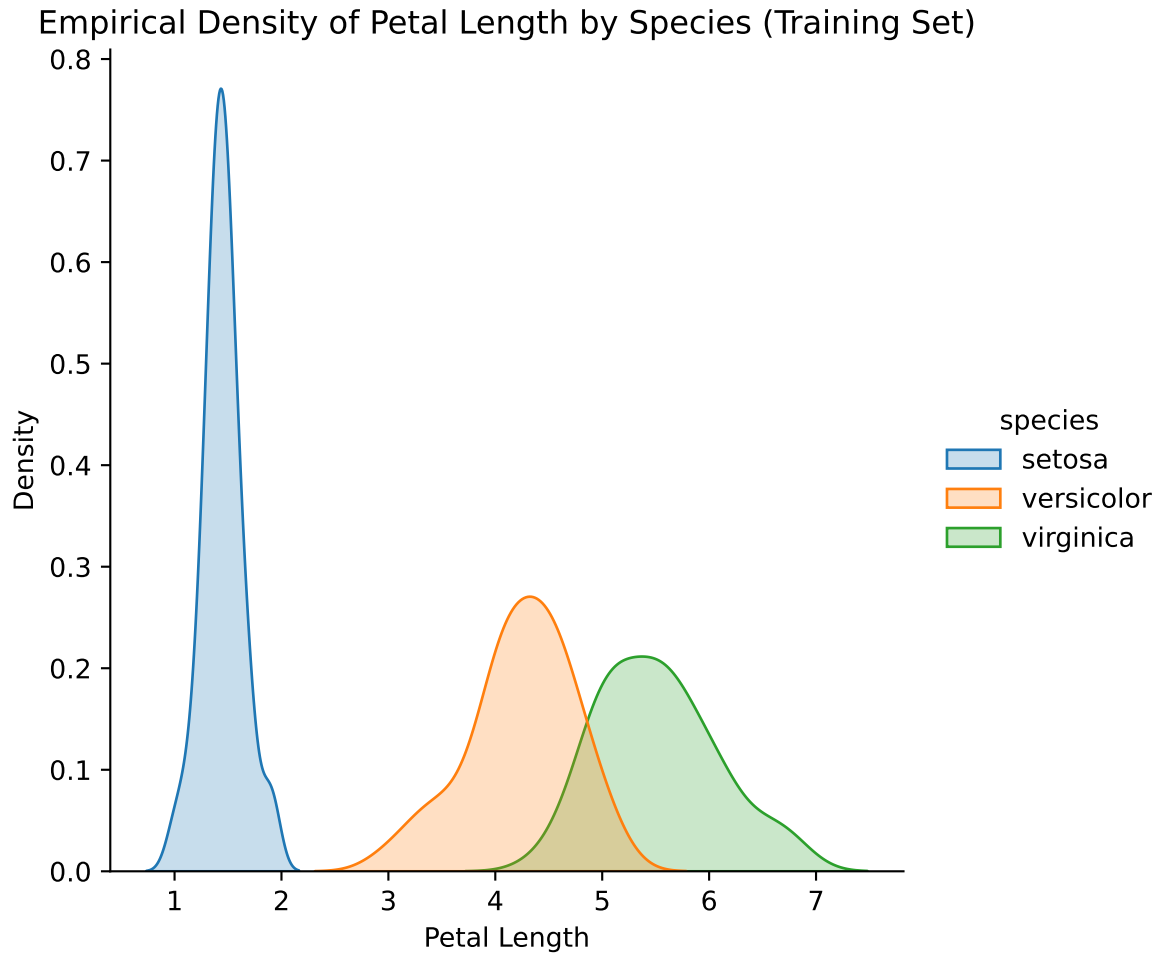
```
print(train_df.columns.tolist())
```

```
['rownames', 'Sepal.Length', 'Sepal.Width', 'Petal.Length', 'Petal.Width', 'Species']
```

```
train_df.columns = (  
    train_df.columns  
    .str.strip()  
    .str.lower()  
    .str.replace(" ", ".", regex=False)  
)  
  
test_df.columns = (  
    test_df.columns  
    .str.strip()  
    .str.lower()  
    .str.replace(" ", ".", regex=False)  
)
```

2(b)

```
sns.displot(  
    data=train_df,  
    x="petal.length",  
    hue="species",  
    kind="kde",  
    fill=True  
)  
  
plt.title("Empirical Density of Petal Length by Species (Training Set)")  
plt.xlabel("Petal Length")  
plt.ylabel("Density")  
plt.show()
```



We plot kernel density estimates using seaborn's KDE, which provides a smooth approximation of the empirical distribution of petal length for each species.

2(c)

```
petal_col = "petal.length"

summary = (
    train_df
    .groupby("species")[petal_col]
    .agg(["count", "mean", "std"])
)
```

```
print(summary)
```

	count	mean	std
species			
setosa	40	1.450000	0.183973
versicolor	41	4.241463	0.481132
virginica	39	5.520513	0.541528

2(d)

```
x = 5.12
petal_col = "petal.length"
likelihoods = {}

for sp in train_df["species"].unique():
    vals = train_df.loc[
        train_df["species"] == sp, petal_col
    ]

    mu = vals.mean()
    sigma = vals.std()

    likelihoods[sp] = norm.pdf(x, loc=mu, scale=sigma)

likelihoods
```

```
{'setosa': np.float64(8.385828085894835e-87),
 'versicolor': np.float64(0.1565430450019126),
 'virginica': np.float64(0.5604136072758437)}
```

Note that each species appears roughly equally often, so $P(y)$ is approximately the same and hence the prior can be ignored in Naives Bayes classification.

```
print("I would classify the species as " + max(likelihoods, key=likelihoods.get) + ".")
```

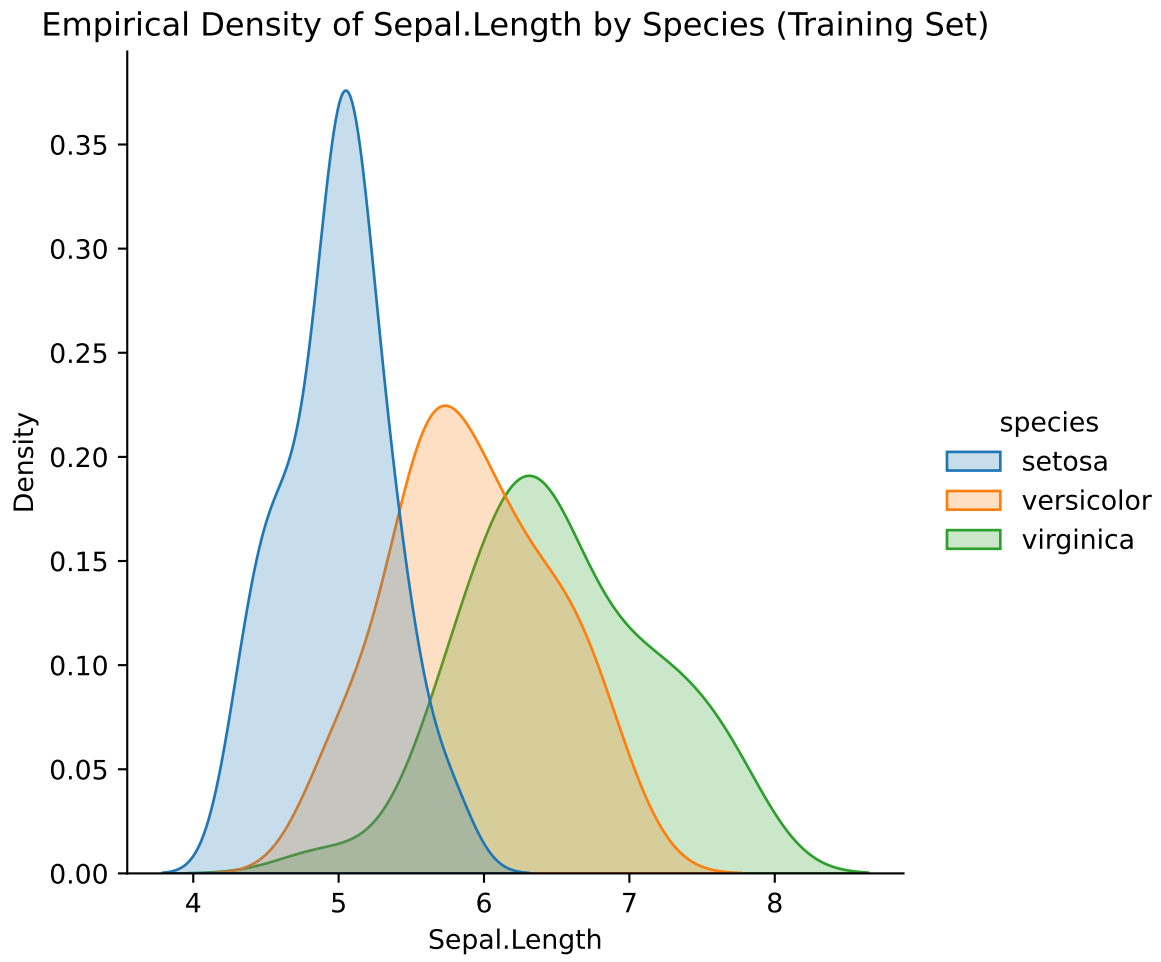
I would classify the species as virginica.

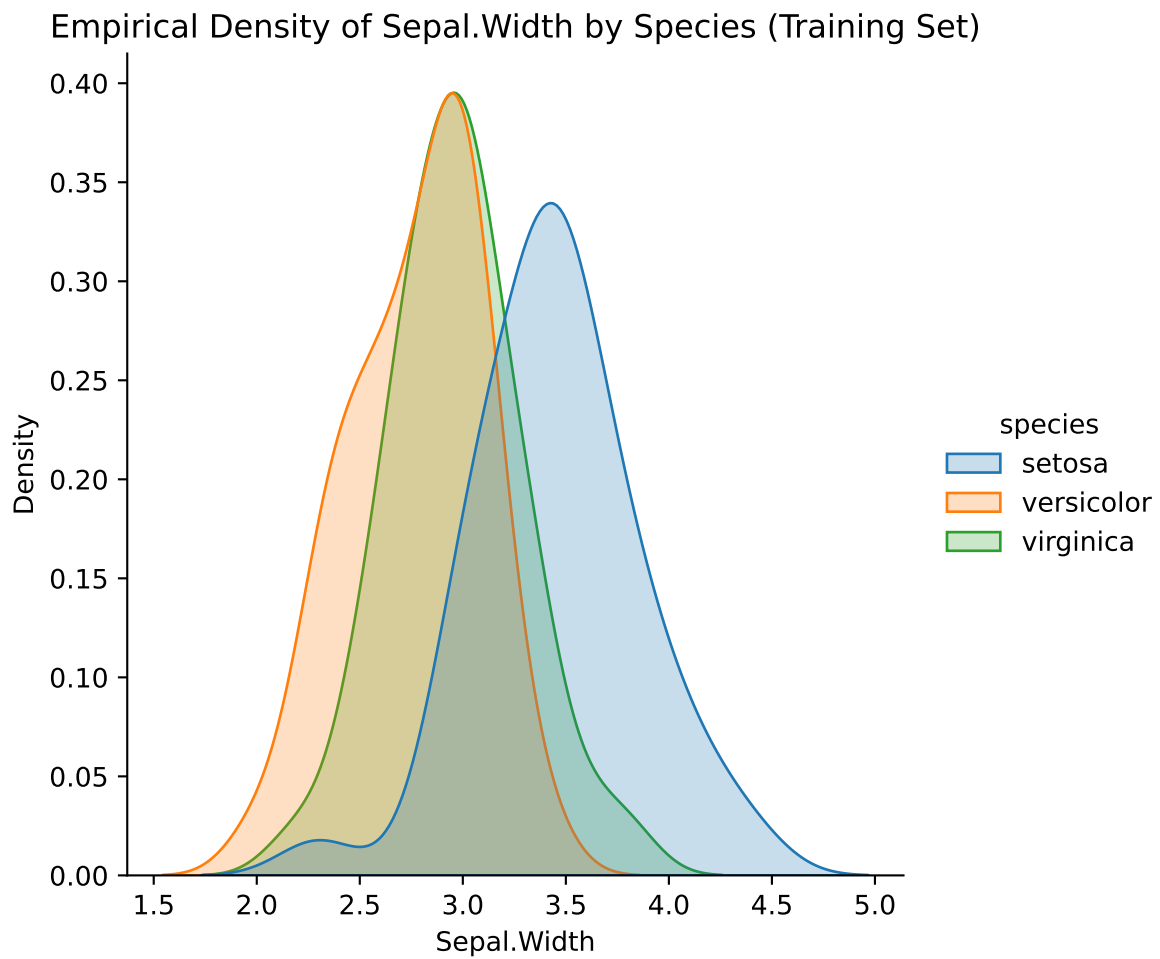
2(e)

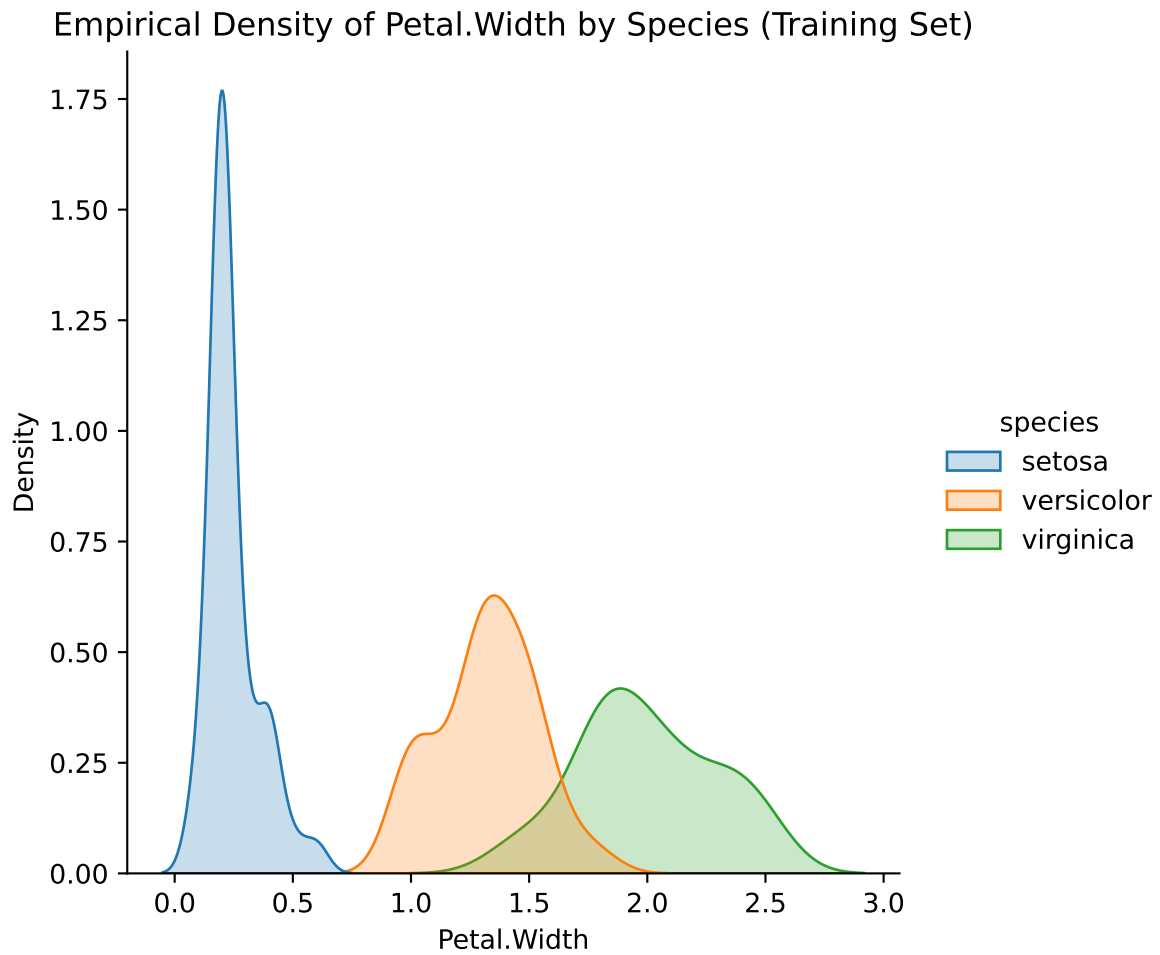
```
features = [
    "sepal.length",
    "sepal.width",
    "petal.width"
]

for feature in features:
    sns.displot(
        data=train_df,
        x=feature,
        hue="species",
        kind="kde",
        fill=True
    )

    plt.title(
        f"Empirical Density of {feature.replace('_', ' ').title()} "
        "by Species (Training Set)"
    )
    plt.xlabel(feature.replace('_', ' ').title())
    plt.ylabel("Density")
    plt.show()
```







2(f)

```
x_new = {  
    "sepal.length": 5.42,  
    "sepal.width": 3.81,  
    "petal.length": 4.23,  
    "petal.width": 2.15  
}  
  
features = list(x_new.keys())  
scores = {}
```

```

for sp in train_df["species"].unique():
    subset = train_df[train_df["species"] == sp]

    log_likelihood = 0

    for feature in features:
        mu = subset[feature].mean()
        sigma = subset[feature].std()

        log_likelihood += np.log(
            norm.pdf(x_new[feature], loc=mu, scale=sigma)
        )

    # equal class priors like (d), so no need to add log(P(sp))
    scores[sp] = log_likelihood

scores

```

```

{'setosa': np.float64(-274.28402662307417),
 'versicolor': np.float64(-13.411970487031656),
 'virginica': np.float64(-8.175213213592475)}

```

```

print("The species is " + max(scores, key=scores.get) + ".")

```

The species is virginica.