DATA 221 Homework 2

Chris Low

Problem 1

1(a)

They are **ordinal categorical variables**. They are categories with a meaningful order (e.g., Worsened less than Unchanged less than Improved). There is a natural order but not we do not have a numerical sense to it.

1(b)

```
import pandas as pd
import statsmodels.api as sm

train_df = pd.read_csv('../hw2/data/PCC_study_train.csv')

# (One-Hot Encoding)
X_cat = pd.get_dummies(train_df[['WBscore', 'PCCsymp']], drop_first=True)
X_cat = X_cat.astype(float)
X_cat = sm.add_constant(X_cat)

y = train_df['vax_status'].map({'Unvaccinated': 0, 'Vaccinated': 1})

# Train Model
log_reg_cat = sm.Logit(y, X_cat).fit()

# View Summary to find significant variables
print(log_reg_cat.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.408057
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:             vax_status   No. Observations:                   75
Model:                          Logit   Df Residuals:                       70
Method:                           MLE   Df Model:                            4
Date:                Wed, 28 Jan 2026   Pseudo R-squ.:                  0.4106
Time:                        23:13:52   Log-Likelihood:                -
30.604
converged:                       True   LL-Null:                       -
51.926
Covariance Type:            nonrobust   LLR p-value:                 1.227e-
08
==============================================================================================
                         coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------------
const                  1.9769      0.521      3.791      0.000       0.955       2.999
WBscore_Unchanged     -2.0144      0.755     -2.668      0.008      -3.494      -
0.535
WBscore_Worsened      -1.0089      1.012     -0.997      0.319      -2.993       0.975
PCCsymp_More          -2.4890      1.069     -2.327      0.020      -4.585      -
0.393
PCCsymp_Same          -2.7364      0.836     -3.275      0.001      -4.374      -
1.099
==============================================================================================
```

If we treat WBscore and PCCsymp as categorical variables, we find that **WBscore_Unchanged, PCCsymp_More, and PCCsymp_Same are statistically significant predictors** of vaccination status, while **WBscore_Worsened** is not significant.

1(c)

Instead of creating many dummy columns, like we did in (b), we are creating two numeric columns, WBscore_num and PCCsymp_num.

```
train_df = pd.read_csv('../hw2/data/PCC_study_train.csv')

y = train_df['vax_status'].map({'Unvaccinated': 0, 'Vaccinated': 1})
```

```python
wb_map  = {'Worsened': 0, 'Unchanged': 1, 'Improved': 2}
pcc_map = {'More': 0, 'Same': 1, 'Less': 2}

train_df['WBscore_num'] = train_df['WBscore'].map(wb_map)
train_df['PCCsymp_num'] = train_df['PCCsymp'].map(pcc_map)

X_num = train_df[['WBscore_num', 'PCCsymp_num']]
X_num = sm.add_constant(X_num)

log_reg_num = sm.Logit(y, X_num).fit()

print(log_reg_num.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.466680
         Iterations 6
                           Logit Regression Results
==============================================================================
Dep. Variable:             vax_status   No. Observations:                   75
Model:                          Logit   Df Residuals:                       72
Method:                           MLE   Df Model:                            2
Date:                Wed, 28 Jan 2026   Pseudo R-squ.:                 0.3259
Time:                        23:13:52   Log-Likelihood:                -
35.001
converged:                       True   LL-Null:                       -
51.926
Covariance Type:            nonrobust   LLR p-value:                 4.462e-
08
==============================================================================
                  coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          -3.6377      0.890     -4.085      0.000      -5.383      -
1.892
WBscore_num     0.9682      0.497      1.948      0.051      -0.006       1.942
PCCsymp_num     1.5780      0.552      2.859      0.004       0.496       2.660
==============================================================================
```

If we treat WBscore and PCCsymp as numeric variables, we find that PCCsymp is a statistically significant predictor of vaccination status, while WBscore is not significant at the 5% level. (Alternatively, we can say that WBscore is marginally significant ($p \approx 0.05$), while PCCsymp is clearly significant.)

1(d)

```
from sklearn.metrics import roc_curve, auc

test_df = pd.read_csv('../hw2/data/PCC_study_test.csv')

y_test = test_df['vax_status2'].map({'Unvaccinated': 0, 'Vaccinated': 1})

# Model from 1(b):
X_test_cat = pd.get_dummies(
    test_df[['WBscore', 'PCCsymp']],
    drop_first=True
)

X_test_cat = X_test_cat.reindex(
    columns=X_cat.columns.drop('const'),
    fill_value=0
)

X_test_cat = sm.add_constant(X_test_cat)
X_test_cat = X_test_cat.astype(float)

# Using train log_reg_cat to predict
y_prob_cat = log_reg_cat.predict(X_test_cat)

fpr_cat, tpr_cat, _ = roc_curve(y_test, y_prob_cat)
auc_cat = auc(fpr_cat, tpr_cat)

print("Categorical AUC:", auc_cat)
```

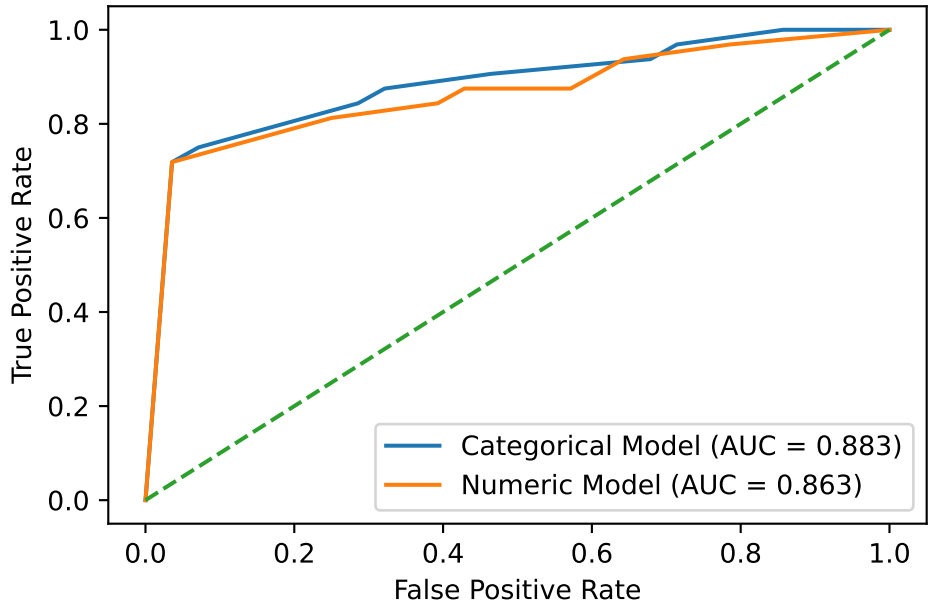
```
Categorical AUC: 0.8828125
```

```
wb_map  = {'Worsened': 0, 'Unchanged': 1, 'Improved': 2}
pcc_map = {'More': 0, 'Same': 1, 'Less': 2}

test_df['WBscore_num'] = test_df['WBscore'].map(wb_map)
test_df['PCCsymp_num'] = test_df['PCCsymp'].map(pcc_map)

X_test_num = test_df[['WBscore_num', 'PCCsymp_num']]
X_test_num = sm.add_constant(X_test_num)
```

```
y_prob_num = log_reg_num.predict(X_test_num)

fpr_num, tpr_num, _ = roc_curve(y_test, y_prob_num)
auc_num = auc(fpr_num, tpr_num)

print("Numerical AUC:", auc_num)
```

```
Numerical AUC: 0.8627232142857143
```

```
import matplotlib.pyplot as plt

plt.figure()
plt.plot(fpr_cat, tpr_cat, label=f'Categorical Model (AUC = {auc_cat:.3f})')
plt.plot(fpr_num, tpr_num, label=f'Numeric Model (AUC = {auc_num:.3f})')
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print("AUC (categorical):", auc_cat)
print("AUC (numeric):", auc_num)
```



```
AUC (categorical): 0.8828125
AUC (numeric): 0.8627232142857143
```

I recommend the **Categorical Model**. It achieves a higher AUC, which means that it captures non-linear relationships between the specific categories (like "Unchanged" vs "Improved") better than the linear assumption of the numeric model.

Problem 2

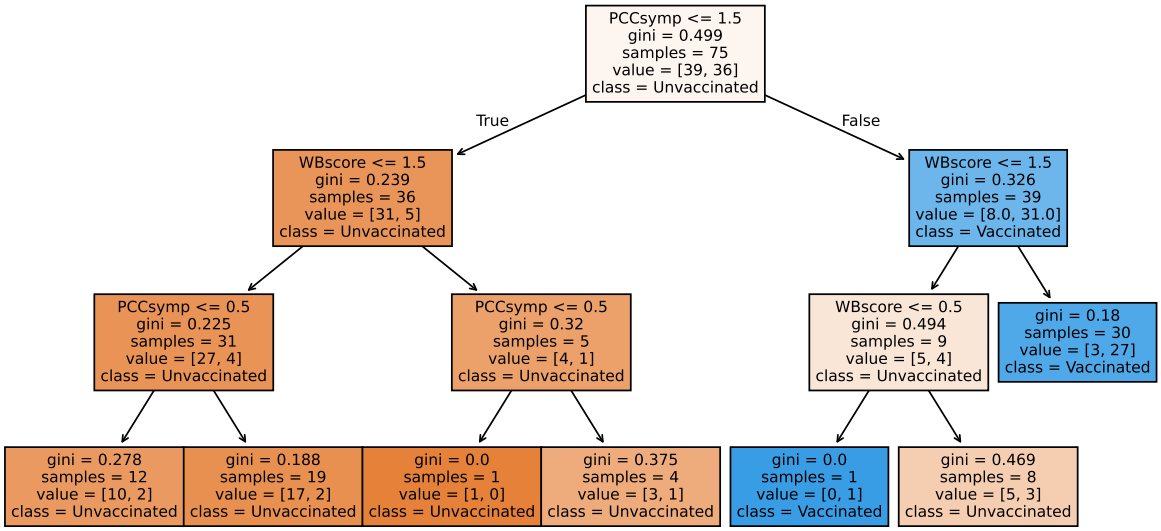
2(a)

```
# Prep data: pasted from above
train_df = pd.read_csv('../hw2/data/PCC_study_train.csv')

y_train = train_df['vax_status'].map({'Unvaccinated': 0, 'Vaccinated': 1})

wb_map  = {'Worsened': 0, 'Unchanged': 1, 'Improved': 2}
pcc_map = {'More': 0, 'Same': 1, 'Less': 2}

train_df['WBscore_num'] = train_df['WBscore'].map(wb_map)
train_df['PCCsymp_num'] = train_df['PCCsymp'].map(pcc_map)

X_train = train_df[['WBscore_num', 'PCCsymp_num']]
```

I will play around with gini and entropy, keeping max_depth at 3 to prevent overfitting.

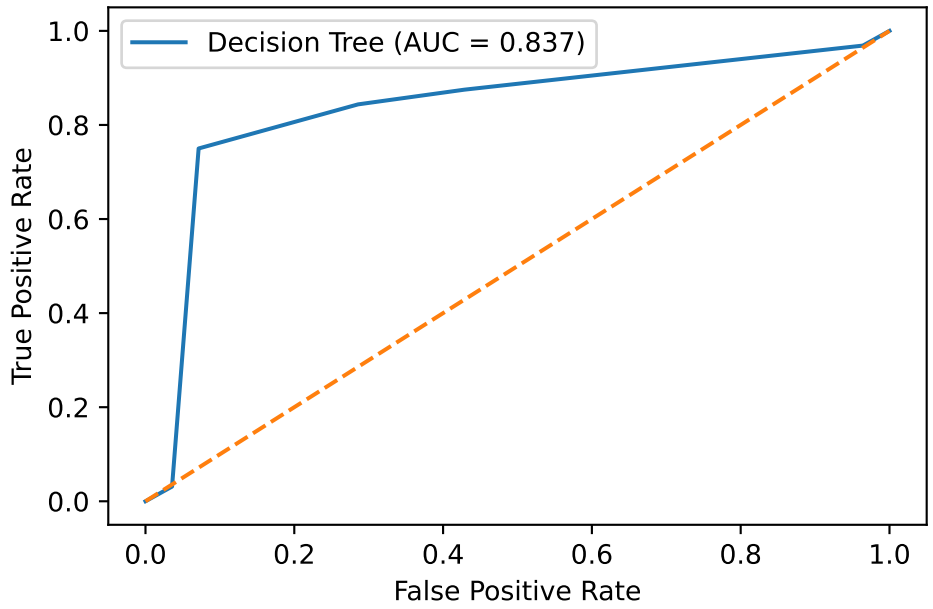
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import plot_tree

tree = DecisionTreeClassifier(
    criterion='gini',
    max_depth=3,
    random_state=42
)

tree.fit(X_train, y_train)

plt.figure(figsize=(12,6))
plot_tree(
```

```
    tree,
    feature_names=['WBscore', 'PCCsymp'],
    class_names=['Unvaccinated', 'Vaccinated'],
    filled=True
)
plt.show()
```




```
test_df = pd.read_csv('../hw2/data/PCC_study_test.csv')

y_test = test_df['vax_status2'].map({'Unvaccinated': 0, 'Vaccinated': 1})

test_df['WBscore_num'] = test_df['WBscore'].map(wb_map)
test_df['PCCsymp_num'] = test_df['PCCsymp'].map(pcc_map)

X_test = test_df[['WBscore_num', 'PCCsymp_num']]

y_prob_tree = tree.predict_proba(X_test)[:, 1]

fpr_tree, tpr_tree, _ = roc_curve(y_test, y_prob_tree)
auc_tree = auc(fpr_tree, tpr_tree)

plt.figure()
plt.plot(fpr_tree, tpr_tree, label=f'Decision Tree (AUC = {auc_tree:.3f})')
plt.plot([0,1], [0,1], linestyle='--')
```

```
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print("Decision Tree AUC:", auc_tree)
```

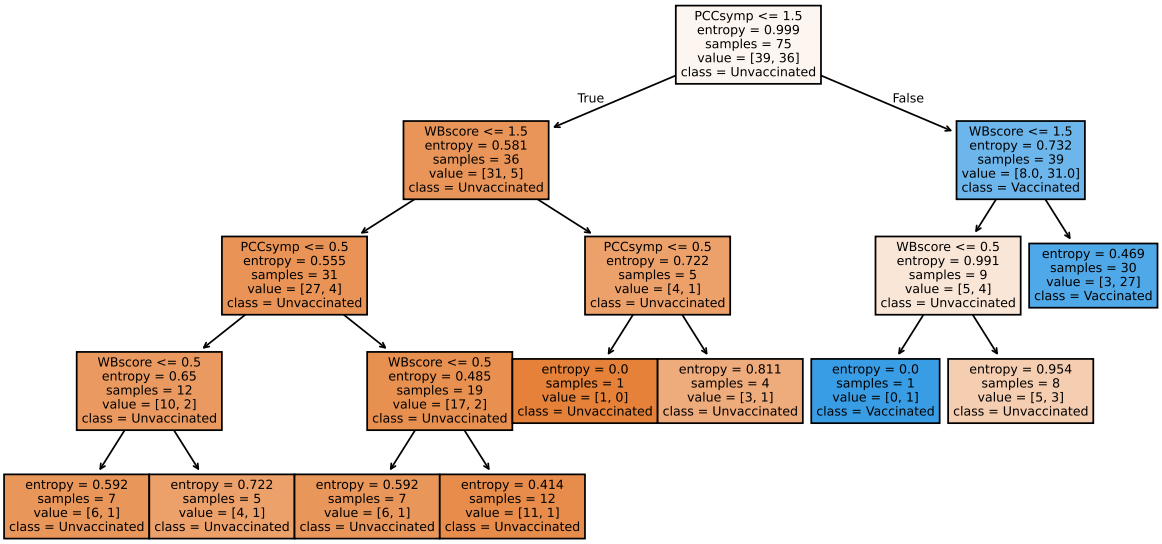


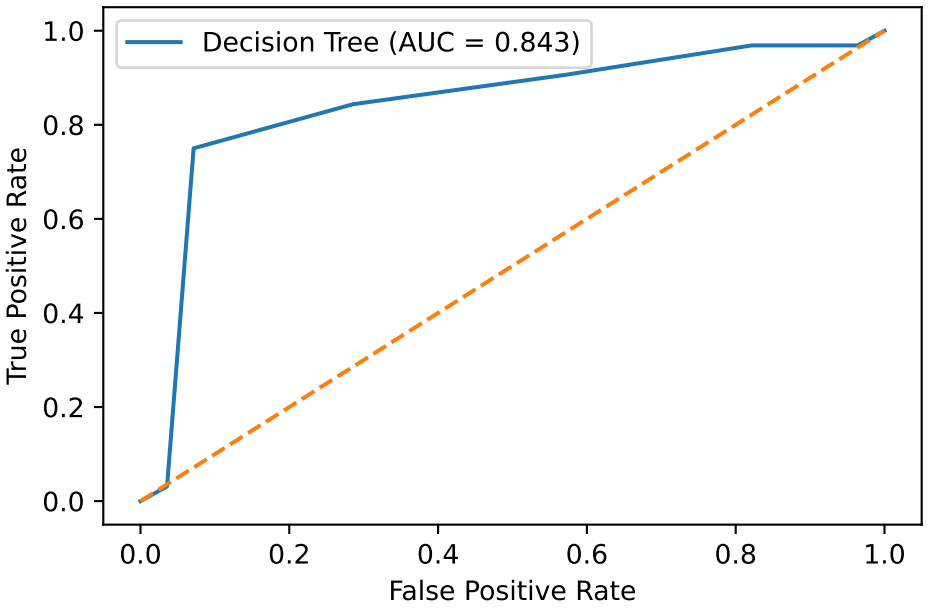

```
Decision Tree AUC: 0.8370535714285715
```

Now I will try entropy with max_depth = 5:

```
dt_model = DecisionTreeClassifier(criterion='entropy', max_depth=5, random_state=42)
dt_model.fit(X_train, y_train)

plt.figure(figsize=(12,6))
plot_tree(
    dt_model,
    feature_names=['WBscore', 'PCCsymp'],
    class_names=['Unvaccinated', 'Vaccinated'],
    filled=True
)
plt.show()
```

```
test_df = pd.read_csv('../hw2/data/PCC_study_test.csv')

y_test = test_df['vax_status2'].map({'Unvaccinated': 0, 'Vaccinated': 1})

test_df['WBscore_num'] = test_df['WBscore'].map(wb_map)
test_df['PCCsymp_num'] = test_df['PCCsymp'].map(pcc_map)

X_test = test_df[['WBscore_num', 'PCCsymp_num']]

y_prob_tree = dt_model.predict_proba(X_test)[:, 1]

fpr_tree, tpr_tree, _ = roc_curve(y_test, y_prob_tree)
auc_tree = auc(fpr_tree, tpr_tree)

plt.figure()
plt.plot(fpr_tree, tpr_tree, label=f'Decision Tree (AUC = {auc_tree:.3f})')
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print("Decision Tree AUC (Entropy):", auc_tree)
```






```
Decision Tree AUC (Entropy): 0.8431919642857144
```

We trained a decision tree classifier using the entropy criterion with a maximum depth of 5. The tree shows that symptom improvement (PCCsymp) is the most important predictor of vaccination status.

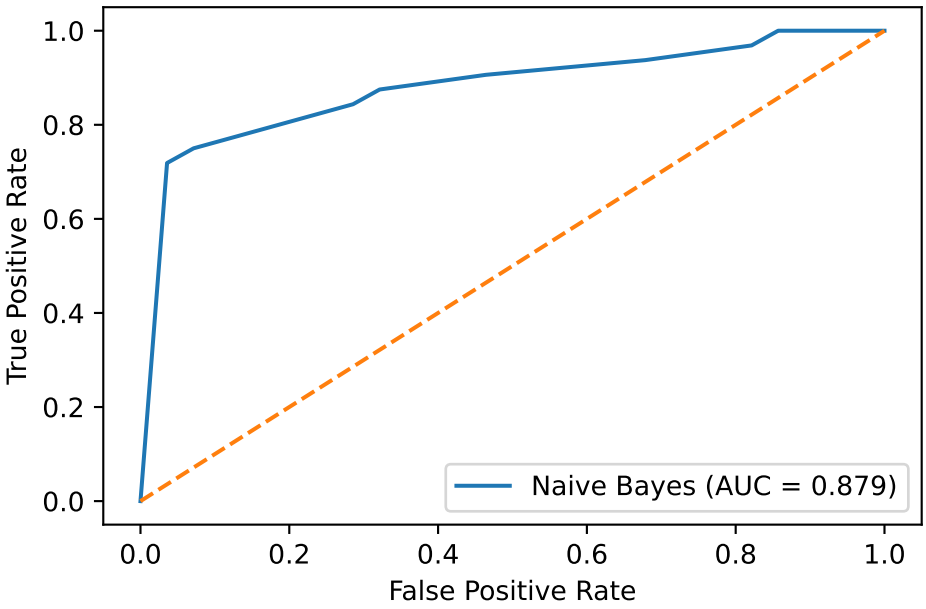
Using predicted probabilities on the test set, the ROC curve yields an AUC of 0.843, which shows a strong classification performance.

2(b)

```python
from sklearn.naive_bayes import CategoricalNB

train_df = pd.read_csv('../hw2/data/PCC_study_train.csv')
test_df  = pd.read_csv('../hw2/data/PCC_study_test.csv')

y_train = train_df['vax_status'].map({'Unvaccinated': 0, 'Vaccinated': 1})
y_test  = test_df['vax_status2'].map({'Unvaccinated': 0, 'Vaccinated': 1})

wb_map  = {'Worsened': 0, 'Unchanged': 1, 'Improved': 2}
pcc_map = {'More': 0, 'Same': 1, 'Less': 2}
```

```
train_df['WBscore_num'] = train_df['WBscore'].map(wb_map)
train_df['PCCsymp_num'] = train_df['PCCsymp'].map(pcc_map)

test_df['WBscore_num'] = test_df['WBscore'].map(wb_map)
test_df['PCCsymp_num'] = test_df['PCCsymp'].map(pcc_map)

X_train = train_df[['WBscore_num', 'PCCsymp_num']]
X_test  = test_df[['WBscore_num', 'PCCsymp_num']]

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

y_prob_nb = nb.predict_proba(X_test)[:, 1]
```

```
fpr_nb, tpr_nb, _ = roc_curve(y_test, y_prob_nb)
auc_nb = auc(fpr_nb, tpr_nb)

plt.figure()
plt.plot(fpr_nb, tpr_nb, label=f'Naive Bayes (AUC = {auc_nb:.3f})')
plt.plot([0,1], [0,1], linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend()
plt.show()

print("Naive Bayes AUC:", auc_nb)
```




```
Naive Bayes AUC: 0.8794642857142857
```

After we trained a Categorical Naive Bayes classifier to predict vaccination status, the predicted probabilities on the test set were used to construct an ROC curve, and the resulting AUC was approximately 0.8795.

2(c)

The AUC Logistic (categorical) ≈ 0.883, AUC Logistic (numeric) ≈ 0.863, the entropy decision tree givce AUC Decision Tree ≈ 0.843, and AUC Naive Bayes ≈ 0.8795.

The **categorical logistic regression model** performs best on this dataset, with the highest test AUC of approximately 0.883.

Treating the survey responses as separate categories appears to capture the relationship with vaccination status more effectively than imposing a numeric structure. The Naive Bayes model performs similarly, which is not surprising given the small sample size and categorical nature of the features. The decision tree shows slightly lower performance, likely due to overfitting and limited generalization from the relatively small training set.

Problem 3

Let y_i be a single observation and let $\hat{p}_i$ denote the predicted probability.

3(a)

The predicted log odds of y_i occurring in terms of $\hat{p}_i$ are

$$\boxed{\log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right)}$$

3(b)

Let the predicted log odds be denoted by ℓ_i, where

$$\ell_i = \log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right).$$

Removing the log:

$$e^{\ell_i} = \frac{\hat{p}_i}{1-\hat{p}_i}.$$

$$e^{\ell_i}(1-\hat{p}_i) = \hat{p}_i.$$

$$e^{\ell_i} - e^{\ell_i}\hat{p}_i = \hat{p}_i \quad \Rightarrow \quad e^{\ell_i} = \hat{p}_i(1+e^{\ell_i}).$$

So,

$$\boxed{\hat{p}_i = \frac{e^{\ell_i}}{1+e^{\ell_i}}}$$

3(c)

Using the same notation, we have

$$\boxed{1-\hat{p}_i = \frac{1}{1+e^{\ell_i}}}$$

3(d)

For likelihood for a Bernoulli variable, we have:

$$L = P(y_i|\hat{p}_i) = \hat{p}_i^{y_i}(1-\hat{p}_i)^{1-y_i}$$

Therefore, for the Bernoulli log likelihood: For a single observation, the Bernoulli log likelihood is

$$\log L = y_i \log(\hat{p}_i) + (1-y_i)\log(1-\hat{p}_i).$$

From (b) and (c), we have:

$$\hat{p}_i = \frac{e^{\ell_i}}{1+e^{\ell_i}}, \qquad 1-\hat{p}_i = \frac{1}{1+e^{\ell_i}}.$$

$$\log L = y_i \log\left(\frac{e^{\ell_i}}{1+e^{\ell_i}}\right) + (1-y_i)\log\left(\frac{1}{1+e^{\ell_i}}\right).$$

$$\log\left(\frac{e^{\ell_i}}{1+e^{\ell_i}}\right) = \ell_i - \log(1+e^{\ell_i}), \qquad \log\left(\frac{1}{1+e^{\ell_i}}\right) = -\log(1+e^{\ell_i}).$$

$$\log L = y_i\left(\ell_i - \log(1+e^{\ell_i})\right) + (1-y_i)\left(-\log(1+e^{\ell_i})\right) = \boxed{y_i\ell_i - \log(1+e^{\ell_i}).}$$

Finally, since $\ell_i = \beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi}$ and from (a) we got $\ell_i = \log\left(\frac{\hat{p}_i}{1-\hat{p}_i}\right)$, we obtain

$$\log L = y_i(\beta_0 + \beta_1 x_{1i} + \cdots + \beta_p x_{pi}) - \log\left(1 + e^{\beta_0+\beta_1 x_{1i}+\cdots+\beta_p x_{pi}}\right),$$

which matches the stated log likelihood.

3(e)

From part (d), for a single observation,

$$\log L = y_i\ell_i - \log(1+e^{\ell_i}).$$

The cross entropy loss is the negative log likelihood:

$$\mathrm{CE}(\ell_i) = -\log L = -y_i\ell_i + \log(1+e^{\ell_i}).$$

Differentiate with respect to ℓ_i:

$$\frac{d}{d\ell_i}\text{CE}(\ell_i) = \frac{d}{d\ell_i}(-y_i\ell_i) + \frac{d}{d\ell_i}\log(1+e^{\ell_i}) = -y_i + \frac{e^{\ell_i}}{1+e^{\ell_i}}.$$

Finally, from (b), we know that $\hat{p}_i = \frac{e^{\ell_i}}{1+e^{\ell_i}}$, so:

$$\frac{d}{d\ell_i}\text{CE}(\ell_i) = \hat{p}_i - y_i = -(y_i - \hat{p}_i),$$

which is the negative residual (residual is defined as $y_i - \hat{p}_i$).

3(f)

From part (d), for a single observation,

$$\log L(\ell_i) = y_i\ell_i - \log(1+e^{\ell_i}).$$

From part (e), we differntiate it with ℓ_i:

$$\frac{d}{d\ell_i}\log L = y_i - \frac{e^{\ell_i}}{1+e^{\ell_i}}.$$

Take the second derivative. The y_i term is constant, so we differentiate only the fraction. Using the quotient rule with $f(\ell_i) = e^{\ell_i}$ and $g(\ell_i) = 1+e^{\ell_i}$, so $f'(\ell_i) = e^{\ell_i}$ and $g'(\ell_i) = e^{\ell_i}$):

$$\frac{d}{d\ell_i}\left(\frac{e^{\ell_i}}{1+e^{\ell_i}}\right) = \frac{e^{\ell_i}(1+e^{\ell_i}) - e^{\ell_i}\cdot e^{\ell_i}}{(1+e^{\ell_i})^2} = \frac{e^{\ell_i}}{(1+e^{\ell_i})^2}.$$

Hence,

$$\frac{d^2}{d\ell_i^2}\log L = -\frac{e^{\ell_i}}{(1+e^{\ell_i})^2}.$$

Finally, rewriting it with the predicted probability from part (b), we have $\hat{p}_i = \frac{e^{\ell_i}}{1+e^{\ell_i}}$ and $1-\hat{p}_i = \frac{1}{1+e^{\ell_i}}$:

$$\hat{p}_i(1-\hat{p}_i) = \frac{e^{\ell_i}}{(1+e^{\ell_i})^2},$$

Therefore:

$$\boxed{\frac{d^2}{d\ell_i^2}\log L = -\hat{p}_i(1-\hat{p}_i).}$$

Problem 4

4(a)

$$f(x, y) = 2x^4 + y^4 - x^2 - 3y^2 + 0.56.$$

Taking partial derivatives:

$$\frac{\partial f}{\partial x} = 8x^3 - 2x \quad \text{and} \quad \frac{\partial f}{\partial y} = 4y^3 - 6y.$$

Therefore, the gradient of (f(x,y)) is

$$\boxed{\nabla f(x, y) = (8x^3 - 2x,\ 4y^3 - 6y)}$$

4(b)

```
def f(x, y):
    return 2*x**4 + y**4 - x**2 - 3*y**2 + 0.56

def grad_f(x, y):
    """
    Gradient of f(x,y) = 2x^4 + y^4 - x^2 - 3y^2 + 0.56
    """
    dfdx = 8*x**3 - 2*x
    dfdy = 4*y**3 - 6*y
    return dfdx, dfdy


def gradient_descent(x0, y0, step_size, K):
    """
    Perform K steps of gradient descent starting from (x0, y0)
    using a constant step size.

    Returns: list of (x, y, f(x,y)) tuples.
    """
    x = float(x0)
    y = float(y0)

    path = [(x, y, f(x, y))]
```

```
    for _ in range(K):
        gx, gy = grad_f(x, y)
        x = x - step_size * gx
        y = y - step_size * gy
        path.append((x, y, f(x, y)))

    return path
```

4(c)

```
path_c = gradient_descent(2, 4, 0.01, 15)
final_x, final_y, final_val = path_c[-1]

print("Final location:", (final_x, final_y))
print("Function value:", final_val)
```

```
Final location: (0.6866318316549618, 1.2726109176569957)
Function value: -1.7026185737001298
```

4(d)

```
path_d = gradient_descent(2, 4, 0.02, 15)
x15_d, y15_d, final_val_d = path_d[-1]

print("Final location:", (x15_d, y15_d))
print("Function value:", final_val_d)
```

```
Final location: (0.5522904565878943, -1.1812130644848766)
Function value: -1.7979749880412519
```

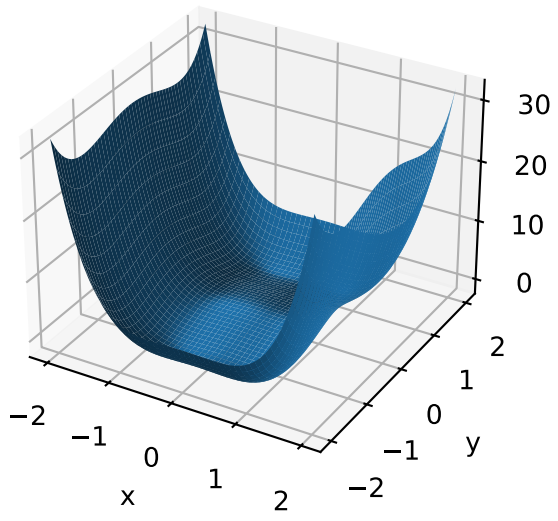
4(e)

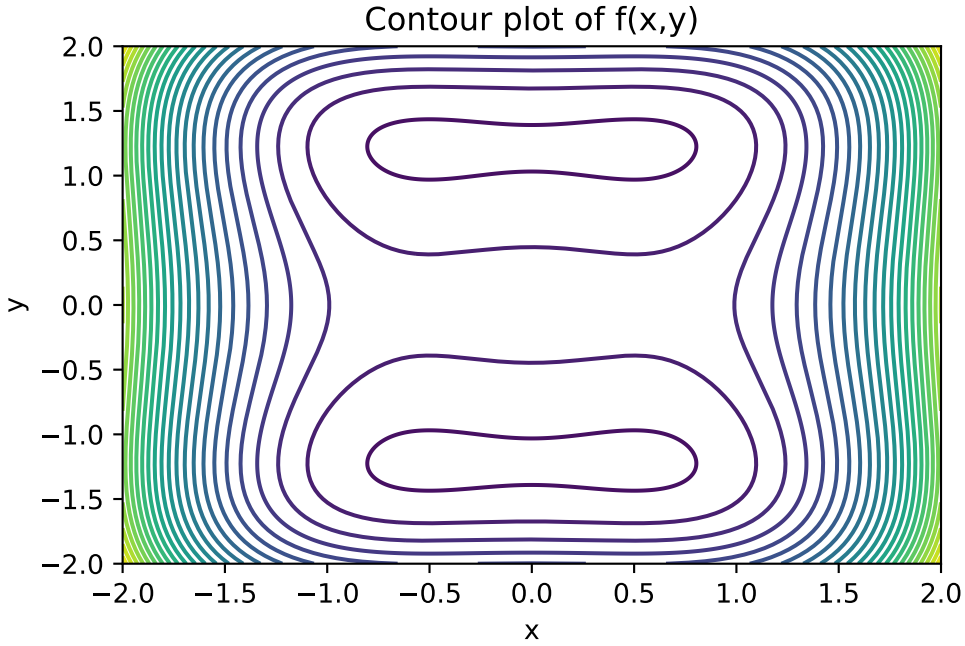
```
path_e = gradient_descent(-1, -1, 0.01, 15)
x15_e, y15_e, final_val_e = path_e[-1]

print("Final location:", (x15_e, y15_e))
print("Function value:", final_val_e)
```

```
Final location: (-0.6429005752849126, -1.1807022893065808)
Function value: -1.7504286747147497
```

4(f)

```python
import numpy as np
import matplotlib.pyplot as plt

def f(x, y):
    return 2*x**4 + y**4 - x**2 - 3*y**2 + 0.56

x = np.linspace(-2, 2, 400)
y = np.linspace(-2, 2, 400)
X, Y = np.meshgrid(x, y)
Z = f(X, Y)

# Surface
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.plot_surface(X, Y, Z)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x,y)')
plt.show()

# Contour
plt.figure()
plt.contour(X, Y, Z, levels=30)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Contour plot of f(x,y)')
plt.show()
```





Interpreation: **(c)** Starting from $(2, 4)$ with a small step size, the algorithm moves gradually and stays in the upper valley of the surface. From the contour plot, it makes sense that it ends near the local minimum with positive (y), since it never crosses into another basin.

(d) When the step size is increased, the updates become larger and the path jumps across the contours near $y = 0$. As a result, the algorithm ends up in a different valley, which explains why it converges to a minimum with negative y.

(e) Starting at $(-1, -1)$, the point is already inside the lower basin shown in the contour plot. Because of this, gradient descent stays in that region and converges to the a similar minimum as in part (d).