Midterm Test

Name: Isabella Blanka ZENKL

Student ID: 25550012

Part I. Binary and Multiple Choices Questions

(A) Binary Choice Questions

A. supervised learning B. unsupervised learning

Use outstanding balance and age to predict whether customers will default on the loan

1. A

Separate customers into different groups based on their purchasing records and gender.

2. B

Use sales revenue from the last quarter to predict sales revenue in next quarter.

3. A

Group all staff into different types based on their job performance.

4. B

Separate credit card transactions into different types based on transaction time and location.

5. B

A. True B. False

Decision Tree models are non-parametric supervised learning method.

6. A

Adding the complexity to a model usually increase its performance on test data.

7. A

Support-Vector Machines (SVMs) can model non-linear relationships with kernel functions.

8. A

Logistic regression is an unregularized model, it doesn't control the coefficient size.

9. B

For SVMs, a smaller C value means stronger regularization on the coefficients.

10. A

(B) Multiple Choice Questions

11. B
12. D
13. D
14. C
15. A
16. C
17. B
18. C
19. D
20. C

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression

np.random.seed(2025)
x = np.random.normal(loc=3,scale=1,size=100)
e = np.random.normal(loc=1,scale=1,size=100)
y = 1 + 2 * x + e

df = pd.DataFrame({"X":x, "y":y})

model = LinearRegression().fit(df[['X']],df['y'])
```

In [2]:
```
display(x)
```

```
array([2.90760982, 3.73428557, 1.56111797, 2.33657798, 2.89927191,
       5.14698657, 4.39370809, 2.79262887, 3.76358729, 3.20505365,
       1.95850383, 2.17192391, 1.94620431, 2.91124134, 3.15819335,
       1.21067695, 4.06348178, 3.03712001, 2.83624207, 2.67179243,
       4.5889765 , 2.81121698, 2.81827703, 3.71790021, 2.99929275,
       3.78216697, 3.08699112, 3.06625557, 4.20056494, 2.93833469,
       2.09909365, 1.74075196, 3.05663286, 2.03378138, 3.74855555,
       3.27818639, 2.68924385, 2.34609501, 2.92705202, 3.59193753,
       3.85760397, 1.80582159, 4.90154073, 2.35481595, 1.81780332,
       4.55220508, 4.21632153, 3.93633718, 2.32444949, 4.0693203 ,
       2.89025798, 1.60676427, 2.32849953, 1.73792511, 3.90213891,
       4.17361397, 1.96070886, 2.8435722 , 3.39459629, 1.83372555,
       3.93706287, 1.99538366, 1.60923089, 2.15372998, 2.62791732,
       2.14049247, 4.2059987 , 3.00188908, 1.57227172, 3.34329041,
       1.56573006, 4.53077249, 1.79794861, 2.55448902, 3.19152721,
       2.22161686, 2.01466251, 3.25274976, 1.43888125, 4.04025353,
       1.70741829, 1.74461719, 2.85518787, 0.7883308 , 2.98211125,
       2.27036558, 3.3332055 , 2.89153769, 3.89915513, 2.10001082,
       3.79592296, 2.99766007, 2.4818247 , 3.21385204, 2.22377355,
       2.4077458 , 2.46053207, 3.17779755, 2.88272135, 0.8680699 ])
```

In [3]:
```
x.shape
```

Out[3]:
```
(100,)
```

21. C
22. A
23. D

Part II. Descriptive Questions

1. Model Training: What is Logistic Regression? Take binary classification as an example, describe how the algorithm find the optimal parameter values and estimate class probabilities?

(max: 200 words)

Logistic regression is a type of classification model that is used to predict a binary outcome. It is a linear model, but it is passed through a logistic function to basically bind it to produce a probability between 0 and 1. To find optimal parameter values and estimate class probabilities it tries to segment data with a straight line by tilting the line in various directions-which represents different parameter (x) values... because the slope is changing.. and thus different relationships between x and y-until it finds the line tilt that best divides the data into classes (according to objective log-loss function... specfically, by trying to reduce log-loss).

2. Model Evaluation: What is overfitting and why it is harmful? How to avoid overfitting in a systematic way?

(max: 200 words)

Overfitting is when a model gets "too good" in its performance with training data, but stops being as good in its performance with testing data. This often happens when a model becomes overly complex: it overly adapts to the conditions it was trained on. It is harmful because the whole point of a model is to be good at predicting an outcome with new data, not to be great with the data it was trained on. To avoid overfitting in a systematic way, one needs to control the complexity of a model. Specifically, one needs to find the sweet spot between complexity and model fit. For mathematical functions we can find that sweet spot by penalizing model complexity.

Part III. Empirical Questions

1. Logistic Regression

A sample of 580 customers whose decision on purchasing a house and $income$ was collected. A researcher uses $income$ to predict customers' decision (y = 1 if the customer will buy, y = 0 if not) with a logistic regression model. The model returns the optimized parameters $w0$ = −1.5 and $w1$ = 0.15, i.e. $f(x)$ = − 1.5 + 0.15 * $income$. Please answer the following questions with Python or manual calculation.

A. What is the probability that a customer will buy a house if his/her $income$ is 20? Round the result to 2 decimal places.

In [4]:
```
income = 20
log_odds = -1.5 + 0.15 * income
display(log_odds)
```

```
1.5
```

In [5]:
```
p = 1 / (1 + np.exp(-log_odds))
print(f"ANSWER A) The probability that a customer will buy a house if his/he
```

```
ANSWER A) The probability that a customer will buy a house if his/her income
is 20 is 0.82
```

B. What is the predicted decision for this customer?

ANSWER B) As the probability they the customer will buy a house is 0.82, and 0.82 > 0.50, then the predicted decision is that they will buy a house.

2. Data Generation and Visualization

A. Create 100 numbers evenly spaced between in the range [0, 10), with step size as 0.1. Save them in a variable named $x1$. (Note: 10 shall not be included.)

In [6]:
```
x1 = np.arange(0, 10, 0.1) #ANSWER A
```

B. Apply natural exponential transformation to $x1$ values, save the transformed values as variable $x2$.

In [7]:
```
x2 = np.exp(x1) #ANSWER B

df = pd.DataFrame({'x1': x1, 'x2': x2})  #ANSWER C

display(df.shape, df.head())
```

```
(100, 2)
```

	x1	x2
0	0.0	1.000000
1	0.1	1.105171
2	0.2	1.221403
3	0.3	1.349859
4	0.4	1.491825

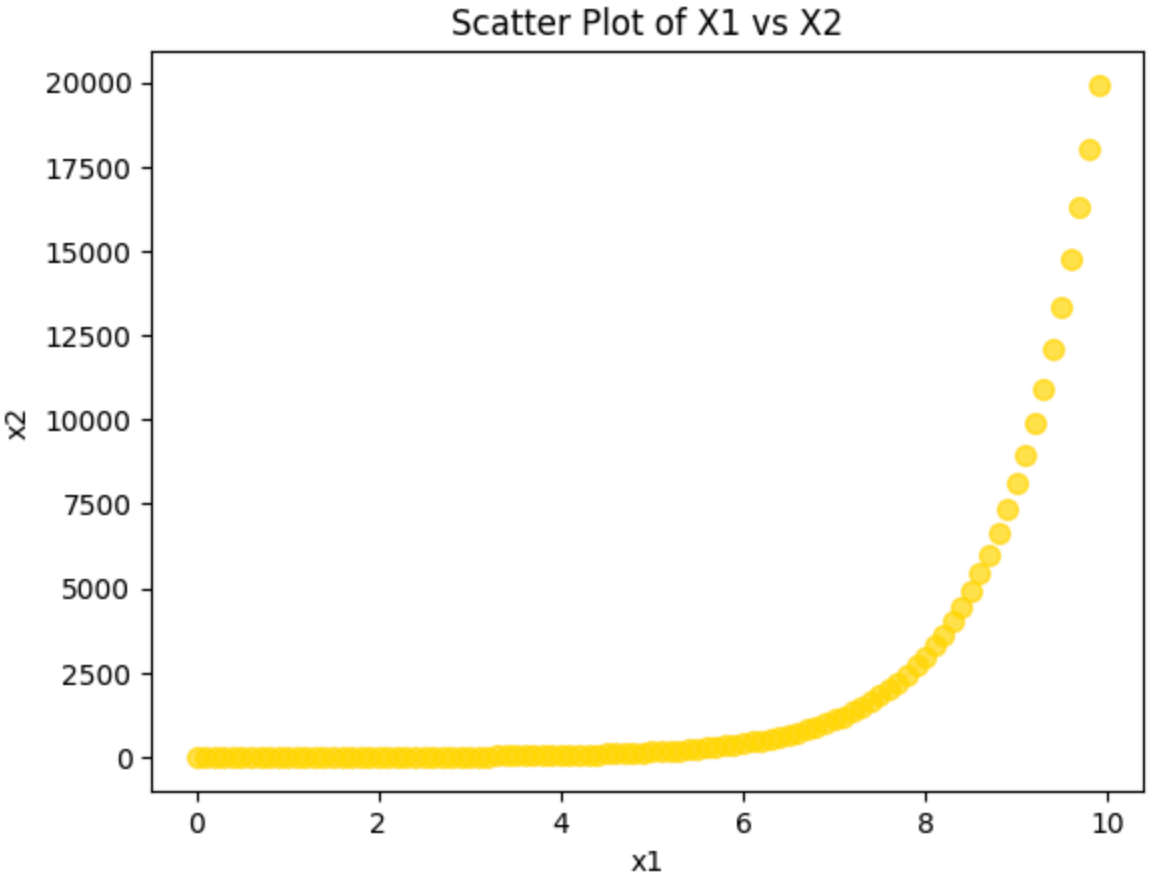
In [8]:
```
print(f"ANSWER C) x1 mean is {x1.mean()} and x2 mean is {round(x2.mean(),2)}
```

```
ANSWER C) x1 mean is 4.95 and x2 mean is 2094.25
```

In [9]:
```
#ANSWER D
import matplotlib.pyplot as plt
plt.scatter(x1, x2, color='gold', alpha=0.7, s=50)

# Add labels and title
plt.xlabel("x1")
plt.ylabel("x2")
plt.title("Scatter Plot of X1 vs X2")

# Display the plot
plt.show()
```



3. SVM: Programming

The Wisconsin Breast Cancer dataset from scikit-learn package contains 30 features and 1 target variable for 569 patients. The features records different characteristics of a patient's tumour, while the target variable refers to the diagnosis: 0 means benign, 1 means malignant.

In [10]:
```
from sklearn.datasets import load_breast_cancer

X, y = load_breast_cancer(return_X_y = True, as_frame = True)

X = X[['mean radius','mean texture','mean perimeter','mean area','mean smoot

display(X.shape, y.shape)
```

```
(569, 5)
(569,)
```

In [11]:
```
X.head()
```

Out[11]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness
0	17.99	10.38	122.80	1001.0	0.11840
1	20.57	17.77	132.90	1326.0	0.08474
2	19.69	21.25	130.00	1203.0	0.10960
3	11.42	20.38	77.58	386.1	0.14250
4	20.29	14.34	135.10	1297.0	0.10030

In [12]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

display(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

print(f"ANSWER A) there are {X_train.shape[0]} instances in the train set an
```

```
(455, 5)
(114, 5)
(455,)
(114,)
ANSWER A) there are 455 instances in the train set and 114 instances in the t
est set
```

In [13]:
```
from sklearn.preprocessing import MinMaxScaler

#ANSWER B) Scale the features with MinMaxScaler

scaler = MinMaxScaler()

X_train_scaled = scaler.fit_transform(X_train)   # combine train and transfo

X_test_scaled  = scaler.transform(X_test)        # apply the scaler on test

print(f"ANSWER B) the minimum value for test features is {round(X_test_scale
```

```
ANSWER B) the minimum value for test features is -0.03 and the maximum value
is 0.88
```

ANSWER B) The minimum value is below 0 as the scaler was fit with the training data. This means that there is data in the test that is outside the training data, which is why is falls below the 0 range. The opposite happens on the upper end, there is higher upper end data on the training data than on the test data, which is why the transformed test values only reach up to 0.88.

In [14]:
```
#ANSWER C) Train 2 linear SVM Models

from sklearn.svm import SVC

svm1 = SVC(kernel='linear', C = 0.1) #SVM Model with C = 0.1

svm1.fit(X_train_scaled, y_train)
```

```
display(svm1.intercept_, svm1.coef_)
```

```
array([2.61882284])
array([[-1.71522678, -1.0012676 , -1.79044312, -1.48753742, -1.05212746]])
```

In [15]:
```
svm2 = SVC(kernel='linear', C = 100) #SVM Model with C = 100

svm2.fit(X_train_scaled, y_train)

display(svm2.intercept_, svm2.coef_, svm2.n_support_)
```

```
array([11.9191538])
array([[ -1.40631046,  -6.87888502,  -9.62625089, -10.35599754,
         -8.81221713]])
array([42, 42], dtype=int32)
```

In [16]:
```
from sklearn.metrics import accuracy_score

train_pred1 = svm1.predict(X_train_scaled)
acc_train1 = round(accuracy_score(train_pred1, y_train),2)
print(f"ANSWER C) The performance of the C=0.1 SVM model on the train data i

train_pred2 = svm2.predict(X_train_scaled)
acc_train2 = round(accuracy_score(train_pred2, y_train),2)
print(f"ANSWER C) The performance of the C=100 SVM model on the train data i
```

```
ANSWER C) The performance of the C=0.1 SVM model on the train data is 0.87
ANSWER C) The performance of the C=100 SVM model on the train data is 0.92
```

In [17]:
```
test_pred1 = svm1.predict(X_test_scaled)
acc_test1 = round(accuracy_score(test_pred1, y_test),2)
print(f"ANSWER C) The performance of the C=0.1 SVM model on the test data is


test_pred2 = svm2.predict(X_test_scaled)
acc_test2 = round(accuracy_score(test_pred2, y_test),2)
print(f"ANSWER C) The performance of the C=100 SVM model on the test data is
```

```
ANSWER C) The performance of the C=0.1 SVM model on the test data is 0.89
ANSWER C) The performance of the C=100 SVM model on the test data is 0.96
```

ANSWER C) Model C=100 is better than model C=0.1 because its performance is better on the test data (0.96 versus 0.6)

In [18]:
```
#ANSWER D)

from sklearn.model_selection import GridSearchCV

svm_linear = SVC(kernel='linear')

range1 = {'C': [0.001, 0.01, 0.1, 1, 10, 50, 100, 150, 200, 500]}    # 10 C

grid1 = GridSearchCV(estimator = svm_linear,      # the model
                     param_grid = range1,         # C values to compare
                     cv = 5)                      # StratifiedKFold without
```

```
grid1.fit(X_train_scaled, y_train)                # search over all C values

print("Best Params: ", grid1.best_params_)                # best C value (w
print("Best cv score: {:.2%}".format(grid1.best_score_))   # average validat
```

```
Best Params:  {'C': 1}
Best cv score: 91.87%
```

(1) The best C value found is C=1

In [19]:
```
pd.DataFrame(grid1.cv_results_)      # convert as dataframe for better reada
```

Out[19]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	params	split0_tes
0	0.001122	0.000414	0.000450	0.000120	0.001	{'C': 0.001}	0.
1	0.000869	0.000020	0.000372	0.000009	0.010	{'C': 0.01}	0.
2	0.000787	0.000020	0.000344	0.000010	0.100	{'C': 0.1}	0.
3	0.000588	0.000029	0.000281	0.000010	1.000	{'C': 1}	0.
4	0.000536	0.000035	0.000247	0.000005	10.000	{'C': 10}	0.
5	0.000660	0.000045	0.000238	0.000007	50.000	{'C': 50}	0.
6	0.000749	0.000068	0.000239	0.000007	100.000	{'C': 100}	0.
7	0.000765	0.000037	0.000238	0.000006	150.000	{'C': 150}	0.
8	0.000818	0.000037	0.000237	0.000006	200.000	{'C': 200}	0.
9	0.001086	0.000075	0.000246	0.000009	500.000	{'C': 500}	0.

(2) The average generalization performance of the best C value in the cross-validation process is 0.92

In [20]:
```
# (3)

svm3 = SVC(kernel='linear', C = 1) #SVM Model with C = 1 (best C value)

svm3.fit(X_train_scaled, y_train)

train_pred3 = svm3.predict(X_train_scaled)
acc_train3 = round(accuracy_score(train_pred3, y_train),2)
print(f"ANSWER (3) The performance of the C=1 SVM model on the train data is

test_pred3 = svm3.predict(X_test_scaled)
acc_test3 = round(accuracy_score(test_pred3, y_test),2)
print(f"ANSWER (3) The performance of the C=1 SVM model on the test (general
```

```
ANSWER (3) The performance of the C=1 SVM model on the train data is 0.92
ANSWER (3) The performance of the C=1 SVM model on the test (generalization)
data is 0.93
```

In [ ]: