Lab exam solution key : Question paper 1

1.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

temp = np.array([62.5, 64.0, 66.2, 55.1, 56.8, 53.7, 68.9, 70.4]).reshape(
    -1, 1
)
score = np.array([78.2, 80.1, 82.7, 70.3, 72.4, 68.9, 85.2, 88.1])

model = LinearRegression()
model.fit(temp, score)

print(
    f"Beta 0 (Intercept): {model.intercept_:.4f}, Beta 1 (Slope): {model.coef_[0]:.4f}"
)

preds = model.predict(temp)
residuals = score - preds
print(f"R2 Score: {r2_score(score, preds):.4f}")
```

2.

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn.linear_model import Lasso, LinearRegression, Ridge
from sklearn.preprocessing import PolynomialFeatures

engine = np.array([1.2, 2.0, 2.5, 1.0, 3.5, 1.4, 1.8, 2.8]).reshape(-1, 1)
co2 = np.array([120, 180, 210, 95, 250, 135, 170, 220])

# Degree 3 Polynomial
poly = PolynomialFeatures(degree=3)
X_p = poly.fit_transform(engine)

poly_model = LinearRegression()
poly_model.fit(X_p, co2)

# Ridge & Lasso on Deg-6
poly6 = PolynomialFeatures(degree=6)
X_p6 = poly6.fit_transform(engine)

ridge = Ridge(alpha=1.0).fit(X_p6, co2)
```

```
lasso = Lasso(alpha=1.0, max_iter=5000).fit(X_p6, co2)

print("Ridge Coeffs:", ridge.coef_)
print("Lasso Coeffs:", lasso.coef_)
```

3.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

X = np.array([[330, 3.8], [300, 3.0], [315, 3.6], [280, 2.8], [340, 3.9]])
y = np.array([1, 0, 1, 0, 1])

# Sklearn fit
model = LogisticRegression()
model.fit(X, y)

preds = model.predict(X)
print("Classification Report:\n", classification_report(y, preds))
```

4.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

X = np.array(
    [
        [-1.2, 0.5],
        [1.1, 0.9],
        [-0.8, 0.3],
        [1.5, 1.2],
        [-1.0, 0.4],
        [0.9, 0.8],
    ]
)
y = np.array([0, 1, 0, 1, 0, 1])

param_grid = {"C": [0.1, 1, 10], "gamma": [0.01, 0.1, 1]}
grid = GridSearchCV(SVC(kernel="rbf"), param_grid, cv=2)
grid.fit(X, y)

print("Best Parameters:", grid.best_params_)
```

5.

```
import numpy as np
```

```
# Gaussian
g_data = np.array([12.4, 11.9, 13.1])
print(f"Gaussian MLE Mean: {np.mean(g_data):.2f}, Var: {np.var(g_data):.2f}")

# Bernoulli
b_data = np.array([1, 0, 1, 0, 1, 0])
print(f"Bernoulli MLE p: {np.mean(b_data):.2f}")

# Poisson
p_data = np.array([27, 5, 2, 5, 2, 7])
print(f"Poisson MLE lambda: {np.mean(p_data):.2f}")
```

6.

```
import numpy as np

X = np.array(
    [
        [2.5, 1.2, 0.5],
        [3.0, 1.5, 0.8],
        [1.8, 0.9, 0.4],
        [2.2, 1.1, 0.6],
        [3.5, 1.8, 1.0],
    ]
)

# Standardize
X_centered = X - np.mean(X, axis=0)
cov_mat = np.cov(X_centered, rowvar=False)

eigvals, eigvecs = np.linalg.eig(cov_mat)
idx = np.argsort(eigvals)[::-1]
components = eigvecs[:, idx[:2]]

X_pca = X_centered @ components
print("Projected Data (Top 2 PCs):\n", X_pca)
```

7.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier

df = pd.DataFrame(
    {
        "Target": [1, 0, 1, 0, 1, 0, 1, 0],
```

```
        "X1": [2.1, 1.8, 2.5, 1.2, 2.8, 1.0, 2.3, 1.5],
        "X2": [0.5, 0.3, 0.7, 0.2, 0.9, 0.1, 0.6, 0.25],
    }
)

X = df[["X1", "X2"]]
y = df["Target"]

clf = DecisionTreeClassifier(max_depth=2, criterion="gini")
clf.fit(X, y)

print("Tree Depth:", clf.get_depth())
print("Training Accuracy:", clf.score(X, y))
```

8.

```
import numpy as np

# Sample data
data = np.array([12.2, 13.0, 11.5, 14.1, 12.8])
prior_mean = 10.0
prior_var = 4.0
sample_var = np.var(data)
n = len(data)

# MAP Formula for Gaussian
sample_mean = np.mean(data)
map_mean = (prior_mean / prior_var + (n * sample_mean) / sample_var) / (
    1 / prior_var + n / sample_var
)

print(f"MLE Mean: {sample_mean:.4f}")
print(f"MAP Mean: {map_mean:.4f}")
```

9.

```
import pandas as pd

# Data creation
df = pd.DataFrame(
    {
        "Weather": ["Sunny", "Cloudy", "Rainy", "Sunny", "Cloudy"],
        "Sprinkler": [1, 0, 0, 1, 0],
        "WetGrass": [1, 1, 1, 0, 0],
    }
)
```

```
# Compute conditional probability table P(WetGrass=1 | Sprinkler)
print(df.groupby("Sprinkler")["WetGrass"].mean())
```

10.

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

X = np.array([[1.0, 2.1], [1.3, 1.9], [5.0, 4.8], [4.8, 5.2], [8.0, 1.0]])

inertias = []
for k in range(1, 4):
    centroids = X[:k]
    for _ in range(10):
        dists = np.linalg.norm(X[:, np.newaxis] - centroids, axis=2)
        labels = np.argmin(dists, axis=1)
        centroids = np.array(
            [
                X[labels == i].mean(axis=0) if len(X[labels == i]) > 0 else centroids[i]
                for i in range(k)
            ]
        )

    inertia = sum(
        np.min(np.linalg.norm(X[:, np.newaxis] - centroids, axis=2), axis=1) ** 2
    )
    inertias.append(inertia)

plt.plot(range(1, 4), inertias, marker="o")
plt.xlabel("k")
plt.ylabel("Inertia")
plt.show()
```