Beginner_level-Question 1

```
import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

import cv2  # For resizing images

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.preprocessing.image import ImageDataGenerator


# Load the CIFAR-10 dataset

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()


# Normalize the images to a range between 0 and 1

train_images, test_images = train_images / 255.0, test_images / 255.0


# Define the class labels for CIFAR-10 dataset

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']


# Focus on 'cat', 'dog', and 'automobile'

focus_classes = ['cat', 'dog', 'automobile']

focus_class_indices = [class_names.index(c) for c in focus_classes]


# Filter training and test data for only these classes

train_mask = np.isin(train_labels, focus_class_indices)

test_mask = np.isin(test_labels, focus_class_indices)

train_images, train_labels = train_images[train_mask.squeeze()], train_labels[train_mask.squeeze()]

test_images, test_labels = test_images[test_mask.squeeze()], test_labels[test_mask.squeeze()]


# Map labels to their new indices (0 for cat, 1 for dog, 2 for car)

train_labels = np.array([focus_class_indices.index(label[0]) for label in train_labels])

test_labels = np.array([focus_class_indices.index(label[0]) for label in test_labels])


# Initialize the ImageDataGenerator for data augmentation

datagen = ImageDataGenerator(

   rotation_range=20,  # Rotate images randomly by up to 20 degrees

   width_shift_range=0.2,  # Shift images horizontally by up to 20%

   height_shift_range=0.2,  # Shift images vertically by up to 20%

   shear_range=0.2,  # Shear images

   zoom_range=0.2,  # Zoom images in or out

   horizontal_flip=True,  # Flip images horizontally

   fill_mode='nearest'  # Fill missing pixels after transformations

)


# Fit the generator to the training data

datagen.fit(train_images)
```

```
# Create the CNN model
model = models.Sequential([
    layers.Input(shape=(32, 32, 3)),  # Input layer with shape (32, 32, 3)
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(128, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    layers.Flatten(),
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.5),  # Dropout layer to prevent overfitting
    layers.Dense(len(focus_classes), activation='softmax')  # Output layer for 3 classes
])

# Compile the model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Train the model using augmented data
history = model.fit(datagen.flow(train_images, train_labels, batch_size=64),
                    epochs=20,
                    validation_data=(test_images, test_labels))

# Function to display images with better clarity
def show_image(img, title="Image"):
    # Resize the image for better clarity (for display purposes only)
    img_resized = cv2.resize(img, (128, 128))  # Resize to a larger size (128x128)
    plt.figure(figsize=(4, 4))
    plt.imshow(img_resized)
    plt.title(title)
    plt.axis('off')
    plt.show()

# Test the model with the first image in the test set
predictions = model.predict(test_images[:1])
predicted_class = np.argmax(predictions[0])

# Display the first image and its predicted class with better clarity
show_image(test_images[0], title=f"Predicted: {focus_classes[predicted_class]}")

# Evaluate the model on test data
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'Test accuracy: {test_acc}')
```

Intermediate_level-Question 1

```
# -*- coding: utf-8 -*-
"""ShadowFox.ipynb

Automatically generated by Colab.

Original file is located at

https://colab.research.google.com/drive/1h9029jJo9Rvs14wiIR_BRxgbIHUSL9jV
"""

from google.colab import files

# Upload the file
uploaded = files.upload()


import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import LabelEncoder

# Step 1: Load the dataset
df = pd.read_csv('Sample - Superstore.csv', encoding='ISO-8859-1')  # Ensure the file name matches

# Step 2: Check the first few rows of the dataset
print(df.head())

# Step 3: Check for missing values
print(df.isnull().sum())

# Step 4: Get a statistical overview of the dataset
print(df.describe())

# Step 5: Data Exploration and Visualization
# Distribution of Sales
plt.figure(figsize=(12, 6))
sns.histplot(df['Sales'], kde=True, color='blue', bins=30)
plt.title('Sales Distribution')
plt.show()

# Distribution of Profit
plt.figure(figsize=(12, 6))
sns.histplot(df['Profit'], kde=True, color='green', bins=30)
plt.title('Profit Distribution')
plt.show()

# Correlation matrix (to see how different numerical features relate to each other)
# Select only numerical columns for correlation
```

```
numeric_columns =
df.select_dtypes(include=['number']).columns

correlation_matrix =
df[numeric_columns].corr()


plt.figure(figsize=(10, 8))

sns.heatmap(correlation_matrix, annot=True,
cmap='coolwarm', fmt='.2f')

plt.title('Correlation Heatmap')

plt.show()


# Step 6: Convert 'Order Date' and 'Ship Date'
to datetime

df['Order Date'] = pd.to_datetime(df['Order
Date'])

df['Ship Date'] = pd.to_datetime(df['Ship
Date'])


# Extract additional time-related features

df['Order_Month'] = df['Order
Date'].dt.month

df['Ship_Month'] = df['Ship Date'].dt.month

df['Order_DayOfWeek'] = df['Order
Date'].dt.dayofweek

df['Ship_DayOfWeek'] = df['Ship
Date'].dt.dayofweek


# Step 7: Convert categorical columns into
numerical values using LabelEncoder

label_encoder = LabelEncoder()

df['Ship Mode'] =
label_encoder.fit_transform(df['Ship Mode'])

df['Segment'] =
label_encoder.fit_transform(df['Segment'])

df['Category'] =
label_encoder.fit_transform(df['Category'])

df['Sub-Category'] =
label_encoder.fit_transform(df['Sub-
Category'])

df['Region'] =
label_encoder.fit_transform(df['Region'])


# Step 8: Define features (X) and target (y)

X = df[['Sales', 'Quantity', 'Discount', 'Ship
Mode', 'Segment', 'Category', 'Sub-Category',
'Region', 'Order_Month', 'Ship_Month',
'Order_DayOfWeek', 'Ship_DayOfWeek']]

y = df['Profit']


# Step 9: Split the data into training and
testing sets

X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2,
random_state=42)


# Step 10: Train a Linear Regression model

model = LinearRegression()

model.fit(X_train, y_train)


# Step 11: Predict on the test set

y_pred = model.predict(X_test)


# Step 12: Evaluate the model

mse = mean_squared_error(y_test, y_pred)

r2 = r2_score(y_test, y_pred)


print(f'Mean Squared Error: {mse}')

print(f'R-squared: {r2}')


# Step 13: Visualize Actual vs Predicted Profit
```

```
plt.figure(figsize=(10, 6))

plt.scatter(y_test, y_pred, color='blue')

plt.plot([y_test.min(), y_test.max()],
[y_test.min(), y_test.max()], color='red', lw=2)

plt.title('Actual vs Predicted Profit')

plt.xlabel('Actual Profit')

plt.ylabel('Predicted Profit')

plt.show()
```