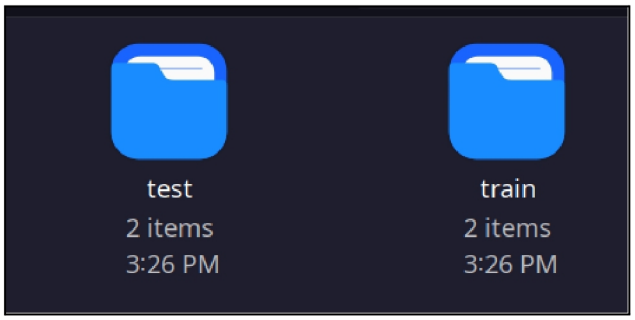
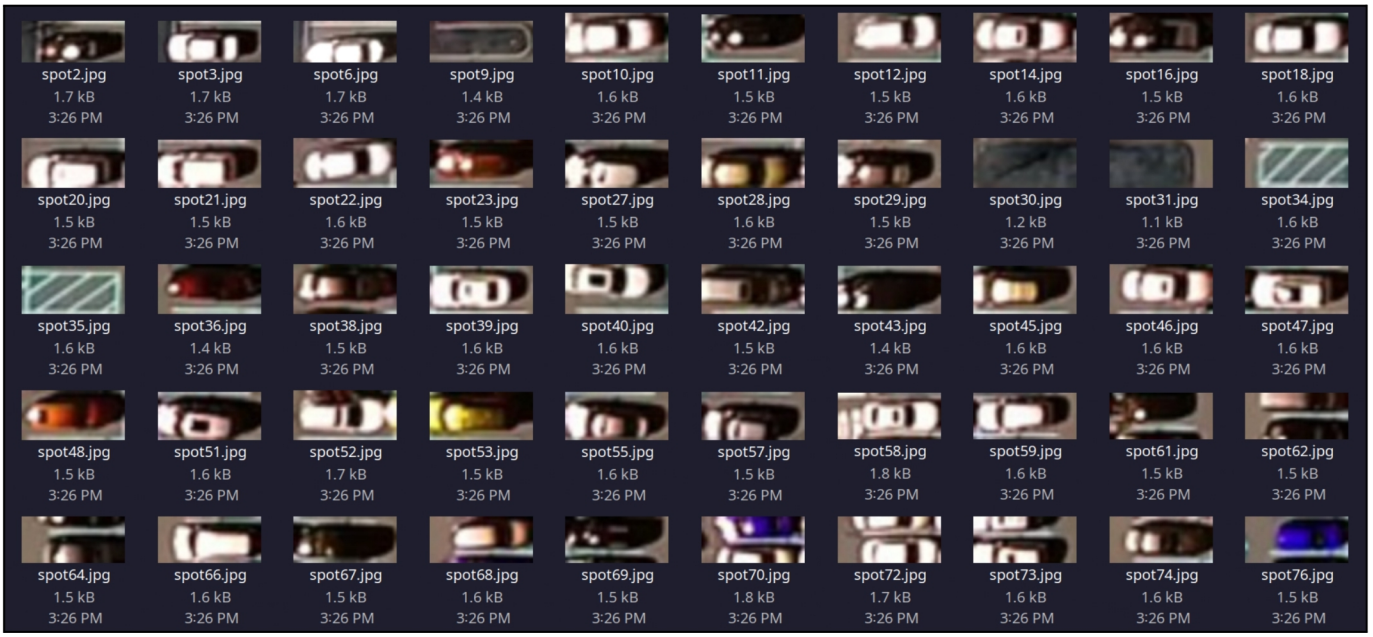
Task 1:Create a Car Parking Prediction Model using TinyML

1. Data Collection
   Collect the required datasets for training and testing the model.

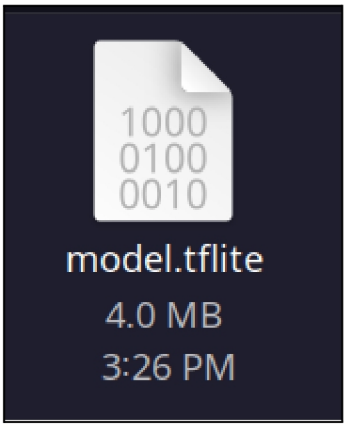




2. Train and prepare the model
   Open and review the `train.ipynb` Jupyter notebook. Understand the data preprocessing, model selection, training process, and evaluation metrics used. Execute the code to train the model.

3. Model File

This directory contains the trained TinyML model file(s). These files are the result of training the model using the training data. Tflite file is generated successfully and are ready for deployment.



4. Checking the Accuracy by Testing Images

This directory contains images used for testing the parking spot detection algorithm. The model is tested for the given input images to check the accuracy

5. Identifying the Parking Spots:

Write the code in `identify_parking_spots.ipynb` Jupyter notebook. It contains code for identifying parking spots in the given images. Understand the algorithms and techniques used for parking spot detection. Execute the code and analyze the results.

```python
from __future__ import division
import matplotlib.pyplot as plt
import cv2
import os, glob
import numpy as np
# from moviepy.editor import VideoFileClip
import tensorflow as tf

# cwd = os.getcwd()
# import tensorflow as tf
```

+ Code + Markdown

```python
interpreter = tf.lite.Interpreter("./model/model.tflite")
interpreter.allocate_tensors()
input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()
```

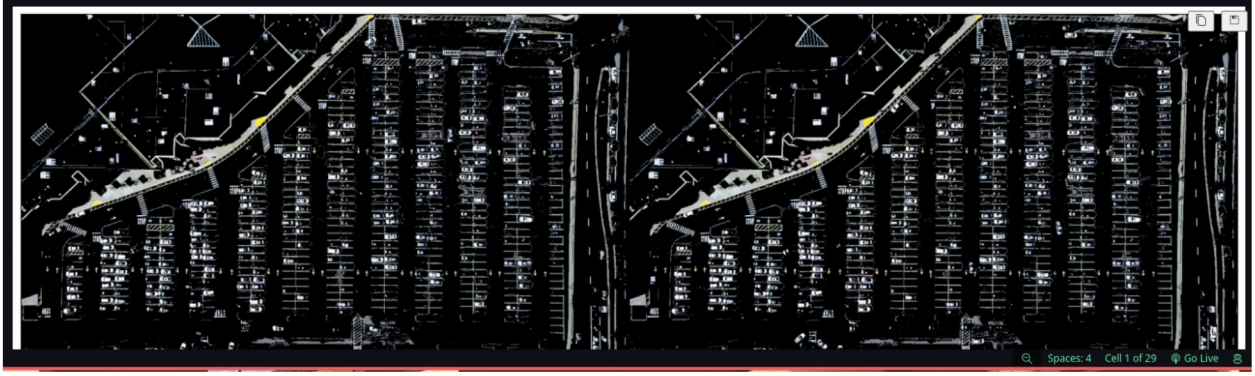
```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

```python
def show_images(images, cmap=None):
    cols = 2
    rows = (len(images) + 1) // cols

    plt.figure(figsize=(15, 12))
    for i, image in enumerate(images):
        plt.subplot(rows, cols, i + 1)
        # use gray scale color map if there is only one channel
        cmap = "gray" if len(image.shape) == 2 else cmap
        plt.imshow(image, cmap=cmap)
        plt.xticks([])
        plt.yticks([])
    plt.tight_layout(pad=0, h_pad=0, w_pad=0)
    plt.show()
```

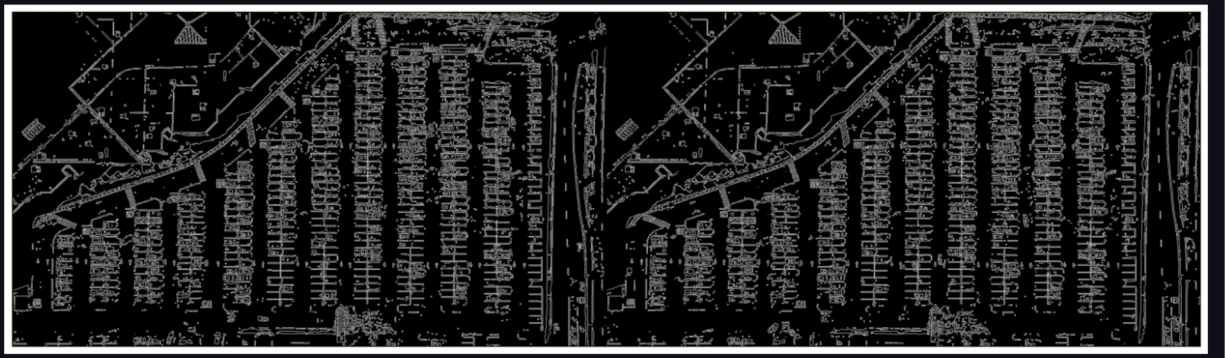
```python
test_images = [plt.imread(path) for path in glob.glob("./test_images/*.jpg")]

show_images(test_images)
```

Color Selection and Edge Detection

```python
# image is expected be in RGB color space# image
def select_rgb_white_yellow(image):
    # white color mask
    lower = np.uint8([120, 120, 120])    Argument of type "list[int]" cannot be assigned to parameter "value" of type "_IntValue" in function "__init__"  Type
    upper = np.uint8([255, 255, 255])    Argument of type "list[int]" cannot be assigned to parameter "value" of type "_IntValue" in function "__init__"  Type
    white_mask = cv2.inRange(image, lower, upper)    No overloads for "inRange" match the provided arguments - (Pylance)
    # yellow color mask
    lower = np.uint8([190, 190, 0])    Argument of type "list[int]" cannot be assigned to parameter "value" of type "_IntValue" in function "__init__"  Type "
    upper = np.uint8([255, 255, 255])    Argument of type "list[int]" cannot be assigned to parameter "value" of type "_IntValue" in function "__init__"  Type
    yellow_mask = cv2.inRange(image, lower, upper)    No overloads for "inRange" match the provided arguments - (Pylance)
    # combine the mask
    mask = cv2.bitwise_or(white_mask, yellow_mask)
    masked = cv2.bitwise_and(image, image, mask=mask)
    return masked


white_yellow_images = list(map(select_rgb_white_yellow, test_images))
show_images(white_yellow_images)
```




```python
def detect_edges(image, low_threshold=50, high_threshold=200):
    return cv2.Canny(image, low_threshold, high_threshold)


edge_images = list(map(lambda image: detect_edges(image), gray_images))

show_images(edge_images)
```



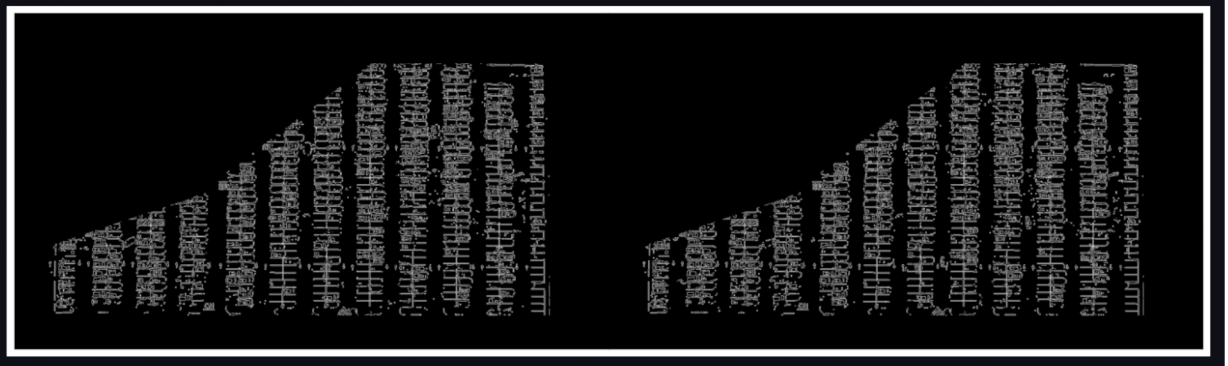
```python
def convert_gray_scale(image):
    return cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)


gray_images = list(map(convert_gray_scale, white_yellow_images))

show_images(gray_images)
```

Python

Identify area of interest

```python
def filter_region(image, vertices):
    """
    Create the mask using the vertices and apply it to the input image
    """
    mask = np.zeros_like(image)
    if len(mask.shape) == 2:
        cv2.fillPoly(mask, vertices, 255)    No overloads for "fillPoly" match the provided arguments - (Pylance)
    else:
        cv2.fillPoly(
            mask, vertices, (255,) * mask.shape[2]
        )  # in case, the input image has a channel dimension
    return cv2.bitwise_and(image, mask)


def select_region(image):
    """
    It keeps the region surrounded by the `vertices` (i.e. polygon).  Other area is set to 0 (black).
    """
    # first, define the polygon by vertices
    rows, cols = image.shape[:2]
    pt_1 = [cols * 0.05, rows * 0.90]
    pt_2 = [cols * 0.05, rows * 0.70]
    pt_3 = [cols * 0.30, rows * 0.55]
    pt_4 = [cols * 0.6, rows * 0.15]
    pt_5 = [cols * 0.90, rows * 0.15]
    pt_6 = [cols * 0.90, rows * 0.90]
    # the vertices are an array of polygons (i.e array of arrays) and the data type must be integer
    vertices = np.array([[pt_1, pt_2, pt_3, pt_4, pt_5, pt_6]], dtype=np.int32)
    return filter_region(image, vertices)


# images showing the region of interest only
roi_images = list(map(select_region, edge_images))

show_images(roi_images)
```

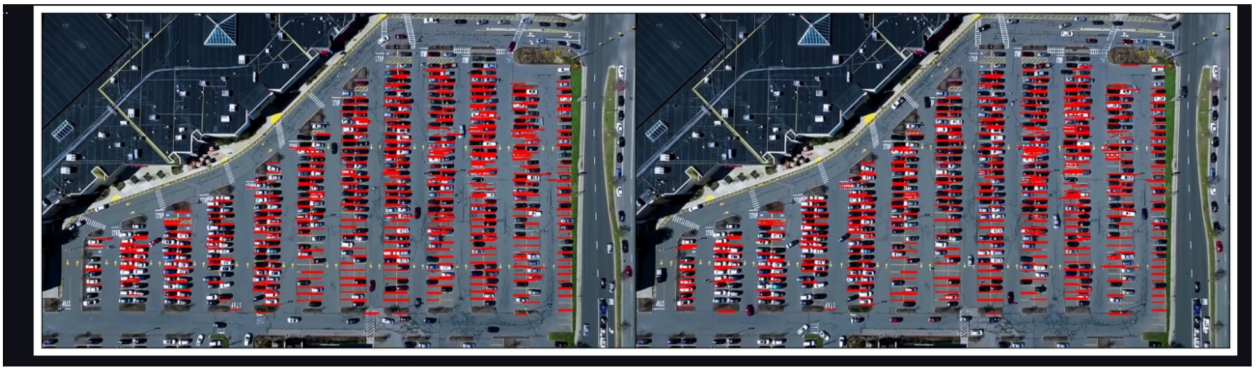
Python



Hough line transform

```python
def hough_lines(image):
    """
    `image` should be the output of a Canny transform.

    Returns hough lines (not the image with lines)
    """
    return cv2.HoughLinesP(
        image, rho=0.1, theta=np.pi / 10, threshold=15, minLineLength=9, maxLineGap=4
    )


list_of_lines = list(map(hough_lines, roi_images))
```

```python
def draw_lines(image, lines, color=[255, 0, 0], thickness=2, make_copy=True):
    # the lines returned by cv2.HoughLinesP has the shape (-1, 1, 4)
    if make_copy:
        image = np.copy(image)  # don't want to modify the original
    cleaned = []
    for line in lines:
        for x1, y1, x2, y2 in line:
            if abs(y2 - y1) <= 1 and abs(x2 - x1) >= 25 and abs(x2 - x1) <= 55:
                cleaned.append((x1, y1, x2, y2))
                cv2.line(image, (x1, y1), (x2, y2), color, thickness)
    print(" No lines detected: ", len(cleaned))
    return image


line_images = []
for image, lines in zip(test_images, list_of_lines):
    line_images.append(draw_lines(image, lines))

show_images(line_images)
```



Identify rectangular blocks of parking

```python
def identify_blocks(image, lines, make_copy=True):
    if make_copy:
        new_image = np.copy(image)
    # Step 1: Create a clean list of lines
    cleaned = []
    for line in lines:
        for x1, y1, x2, y2 in line:
            if abs(y2 - y1) <= 1 and abs(x2 - x1) >= 25 and abs(x2 - x1) <= 55:
                cleaned.append((x1, y1, x2, y2))

    # Step 2: Sort cleaned by x1 position
    import operator

    list1 = sorted(cleaned, key=operator.itemgetter(0, 1))

    # Step 3: Find clusters of x1 close together - clust_dist apart
    clusters = {}
    dIndex = 0
    clus_dist = 10

    for i in range(len(list1) - 1):
        distance = abs(list1[i + 1][0] - list1[i][0])
        #        print(distance)
        if distance <= clus_dist:
            if not dIndex in clusters.keys():
                clusters[dIndex] = []
            clusters[dIndex].append(list1[i])
            clusters[dIndex].append(list1[i + 1])

        else:
            dIndex += 1

```

```python
    # Step 4: Identify coordinates of rectangle around this cluster
    rects = {}
    i = 0
    for key in clusters:
        all_list = clusters[key]
        cleaned = list(set(all_list))
        if len(cleaned) > 5:
            cleaned = sorted(cleaned, key=lambda tup: tup[1])
            avg_y1 = cleaned[0][1]
            avg_y2 = cleaned[-1][1]
            #         print(avg_y1, avg_y2)
            avg_x1 = 0
            avg_x2 = 0
            for tup in cleaned:
                avg_x1 += tup[0]
                avg_x2 += tup[2]
            avg_x1 = avg_x1 / len(cleaned)
            avg_x2 = avg_x2 / len(cleaned)
            rects[i] = (avg_x1, avg_y1, avg_x2, avg_y2)
            i += 1

    print("Num Parking Lanes: ", len(rects))
    # Step 5: Draw the rectangles on the image
    buff = 7
    for key in rects:
        tup_topLeft = (int(rects[key][0] - buff), int(rects[key][1]))
        tup_botRight = (int(rects[key][2] + buff), int(rects[key][3]))
        #         print(tup_topLeft, tup_botRight)
        cv2.rectangle(new_image, tup_topLeft, tup_botRight, (0, 255, 0), 3)
    return new_image, rects


# images showing the region of interest only
rect_images = []
rect_coords = []
for image, lines in zip(test_images, list_of_lines):
    new_image, rects = identify_blocks(image, lines)
    rect_images.append(new_image)
    rect_coords.append(rects)

show_images(rect_images)
```

[13] Python

```
Num Parking Lanes:  12
Num Parking Lanes:  12
```

Identify each spot and count num of parking spaces

Next step-

1. Based on width of each parking line segment into individual spots
2. draw a visualization of all parking spaces

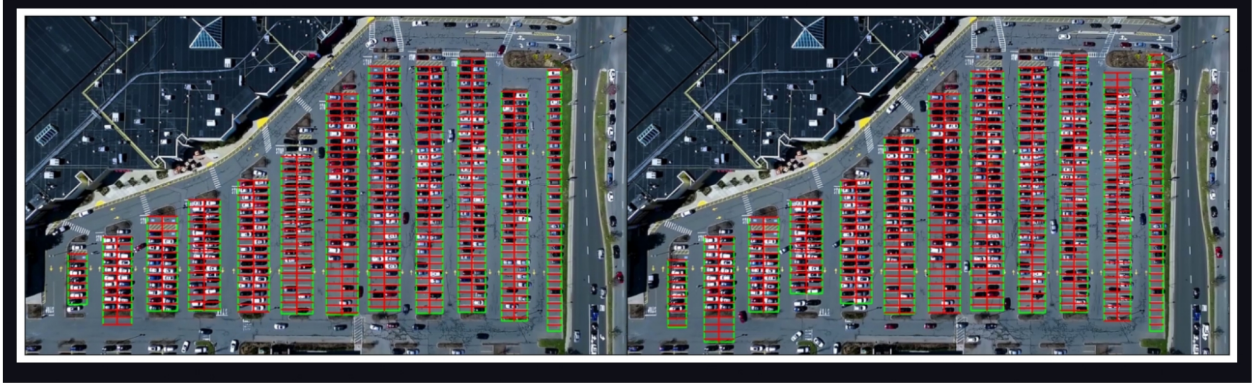
```python
def draw_parking(
    image, rects, make_copy=True, color=[255, 0, 0], thickness=2, save=True
):
    if make_copy:
        new_image = np.copy(image)
    gap = 15.5
    (variable) adj_y1: dict[int, int]  ID to its coords

    adj_y1 = {
        0: 20,
        1: -10,
        2: 0,
        3: -11,
        4: 28,
        5: 5,
        6: -15,
        7: -15,
        8: -10,
        9: -30,
        10: 9,
        11: -32,
    }
```

```
        11: 0,
    }
    for key in rects:
        # Horizontal lines
        tup = rects[key]
        x1 = int(tup[0] + adj_x1[key])
        x2 = int(tup[2] + adj_x2[key])
        y1 = int(tup[1] + adj_y1[key])
        y2 = int(tup[3] + adj_y2[key])
        cv2.rectangle(new_image, (x1, y1), (x2, y2), (0, 255, 0), 2)
        num_splits = int(abs(y2 - y1) // gap)
        for i in range(0, num_splits + 1):
            y = int(y1 + i * gap)
            cv2.line(new_image, (x1, y), (x2, y), color, thickness)
        if key > 0 and key < len(rects) - 1:
            # draw vertical lines
            x = int((x1 + x2) / 2)
            cv2.line(new_image, (x, y1), (x, y2), color, thickness)
        # Add up spots in this lane
        if key == 0 or key == (len(rects) - 1):
            tot_spots += num_splits + 1
        else:
            tot_spots += 2 * (num_splits + 1)

        # Dictionary of spot positions
        if key == 0 or key == (len(rects) - 1):
            for i in range(0, num_splits + 1):
                cur_len = len(spot_dict)
                y = int(y1 + i * gap)
                spot_dict[(x1, y, x2, y + gap)] = cur_len + 1
        else:
            for i in range(0, num_splits + 1):
                cur_len = len(spot_dict)
                y = int(y1 + i * gap)
                x = int((x1 + x2) / 2)
                spot_dict[(x1, y, x, y + gap)] = cur_len + 1
                spot_dict[(x, y, x2, y + gap)] = cur_len + 2

    print("total parking spaces: ", tot_spots, cur_len)
    if save:
        filename = "with_parking.jpg"
        cv2.imwrite(filename, new_image)
    return new_image, spot_dict

```

```
delineated = []
spot_pos = []
for image, rects in zip(test_images, rect_coords):
    new_image, spot_dict = draw_parking(image, rects)
    delineated.append(new_image)
    spot_pos.append(spot_dict)

show_images(delineated)
```

Python

```
total parking spaces:  541 540
total parking spaces:  552 551
```



```
final_spot_dict = spot_pos[1]
```

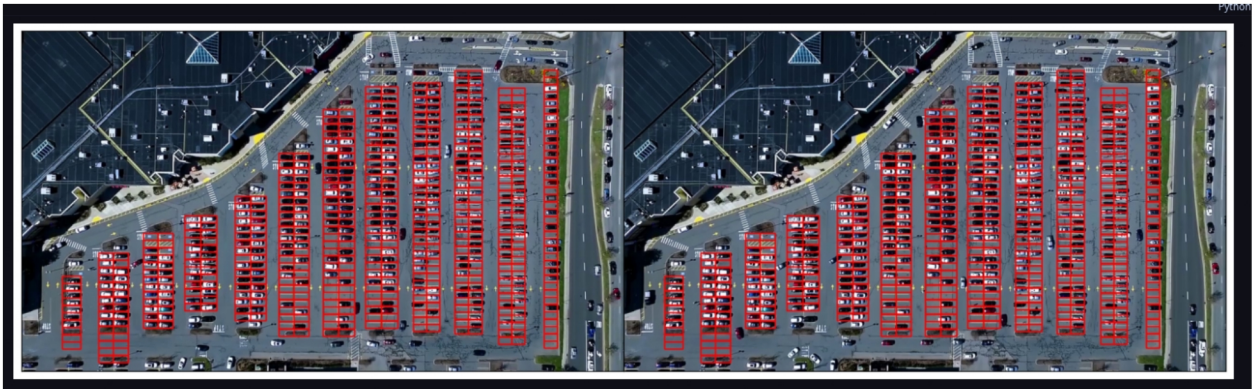
Python

```
print(len(final_spot_dict))
```

Python

```
552
```

```
def assign_spots_map(
    image, spot_dict=final_spot_dict, make_copy=True, color=[255, 0, 0], thickness=2
):
    if make_copy:
        new_image = np.copy(image)
    for spot in spot_dict.keys():
        (x1, y1, x2, y2) = spot
        cv2.rectangle(
            new_image, (int(x1), int(y1)), (int(x2), int(y2)), color, thickness
        )
    return new_image


marked_spot_images = list(map(assign_spots_map, test_images))
show_images(marked_spot_images)
```



```python
# Save spot dictionary as pickle file
import pickle

with open("spot_dict.pickle", "wb") as handle:
    pickle.dump(final_spot_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

Save image for CNN model

```python
def save_images_for_cnn(image, spot_dict=final_spot_dict, folder_name="for_cnn"):
    for spot in spot_dict.keys():
        (x1, y1, x2, y2) = spot
        (x1, y1, x2, y2) = (int(x1), int(y1), int(x2), int(y2))
        # crop this image
        #         print(image.shape)
        spot_img = image[y1:y2, x1:x2]
        spot_img = cv2.resize(spot_img, (0, 0), fx=2.0, fy=2.0)
        spot_id = spot_dict[spot]

        filename = "spot" + str(spot_id) + ".jpg"
        print(spot_img.shape, filename, (x1, x2, y1, y2))

        cv2.imwrite(os.path.join(folder_name, filename), spot_img)


# save_images_for_cnn(test_images[0])
```

[19]

Use trained CNN model to make predictions

```python
from PIL import Image

# model = load_model(top_model_weights_path)
```

[20]

```python
def make_prediction(img):
    # Rescale image
    # img = image/255.

    # #Convert to a 4D tensor
    # image = np.expand_dims(img, axis=0)
    # #print(image.shape)

    # # make predictions on the preloaded model
    # class_predicted = model.predict(image)
    # inID = np.argmax(class_predicted[0])
    # label = class_dictionary[inID]
    # return label

    new_img = cv2.resize(img, (224, 224))

    # input_details[0]['index'] = the index which accepts the input
    interpreter.set_tensor(input_details[0]["index"], [new_img])

    # run the inference
    interpreter.invoke()

    # output_details[0]['index'] = the index which provides the input
    output_data = interpreter.get_tensor(output_details[0]["index"])

    return "empty" if output_data[0][0] > output_data[0][1] else "occupied"
```

[21]

```python
from tqdm import tqdm
```

```python

def predict_on_image(
    image, spot_dict=final_spot_dict, make_copy=True, color=[0, 255, 0], alpha=0.5
):
    if make_copy:
        new_image = np.copy(image)
        overlay = np.copy(image)
    cnt_empty = 0
    all_spots = 0
    for spot in tqdm(spot_dict.keys()):
        all_spots += 1
        (x1, y1, x2, y2) = spot
        (x1, y1, x2, y2) = (int(x1), int(y1), int(x2), int(y2))
        # crop this image
        spot_img = image[y1:y2, x1:x2]
        spot_img = cv2.resize(spot_img, (48, 48))

        label = make_prediction(spot_img)
        #        print(label)
        if label == "empty":
            cv2.rectangle(overlay, (int(x1), int(y1)), (int(x2), int(y2)), color, -1)
            cnt_empty += 1

    cv2.addWeighted(overlay, alpha, new_image, 1 - alpha, 0, new_image)

    cv2.putText(
        new_image,
        "Available: %d spots" % cnt_empty,
        (30, 95),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.7,
        (255, 255, 255),
        2,
    )

    cv2.putText(
        new_image,
        "Total: %d spots" % all_spots,
        (30, 125),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.7,
        (255, 255, 255),
        2,
    )
```

```python
    cv2.putText(
        new_image,
        "Total: %d spots" % all_spots,
        (30, 125),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.7,
        (255, 255, 255),
        2,
    )
    save = False

    if save:
        filename = "with_marking.jpg"
        cv2.imwrite(filename, new_image)

    return new_image


predicted_images = list(map(predict_on_image, test_images))
show_images(predicted_images)
```

```
[22]
...   1%|          | 3/552 [00:00<00:24, 22.30it/s]
    100%|██████████| 552/552 [00:19<00:00, 28.27it/s]
    100%|██████████| 552/552 [00:19<00:00, 27.84it/s]
```

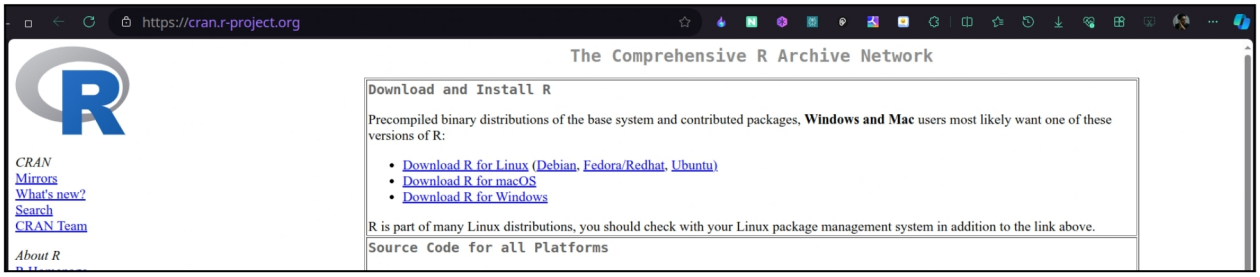
6. Output Image - “with_parking.jpg” :
This is an output image showing parking spots identified or predicted by the algorithm. It has the identified the empty slots where the car can be parked.

By following these steps, you can build a TinyML-based car parking prediction project. You'll be able to assess the effectiveness of the parking spot detection algorithm and the trained TinyML model.

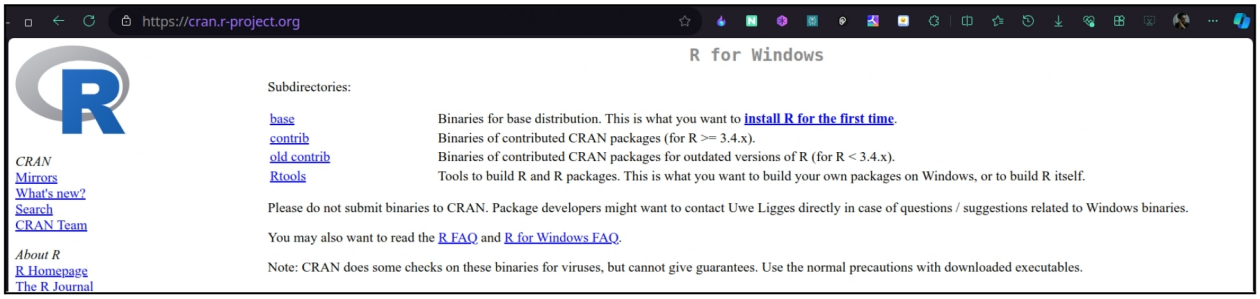
Task 2: Data Analytics using R programming

Setting up R and RStudio -

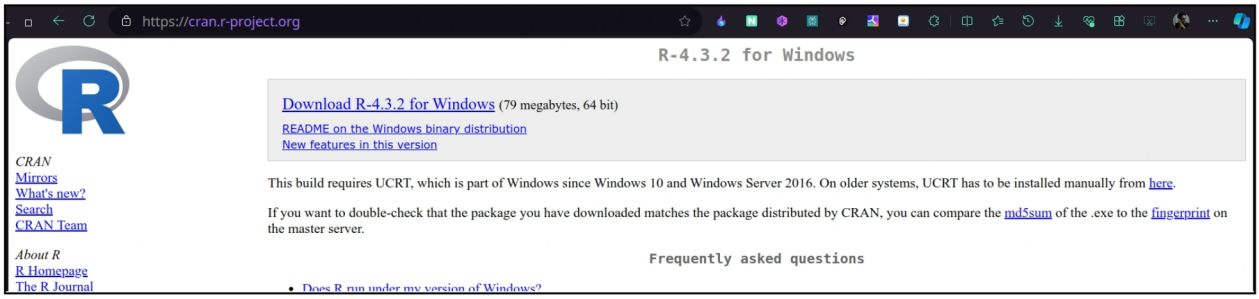
1. To install R, go to CRAN . Click Download R for Windows.



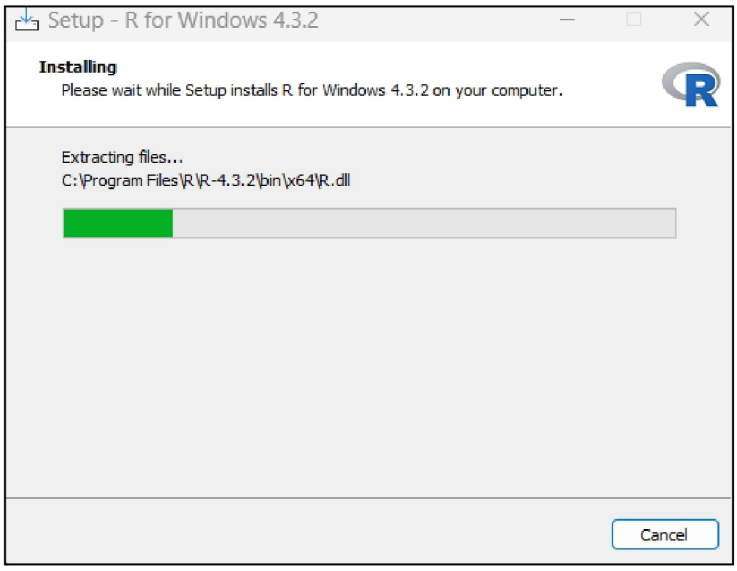
2. Install R Click on install R for the first time.



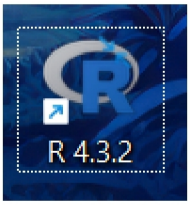
3. Click Download R for Windows. Open the downloaded file.



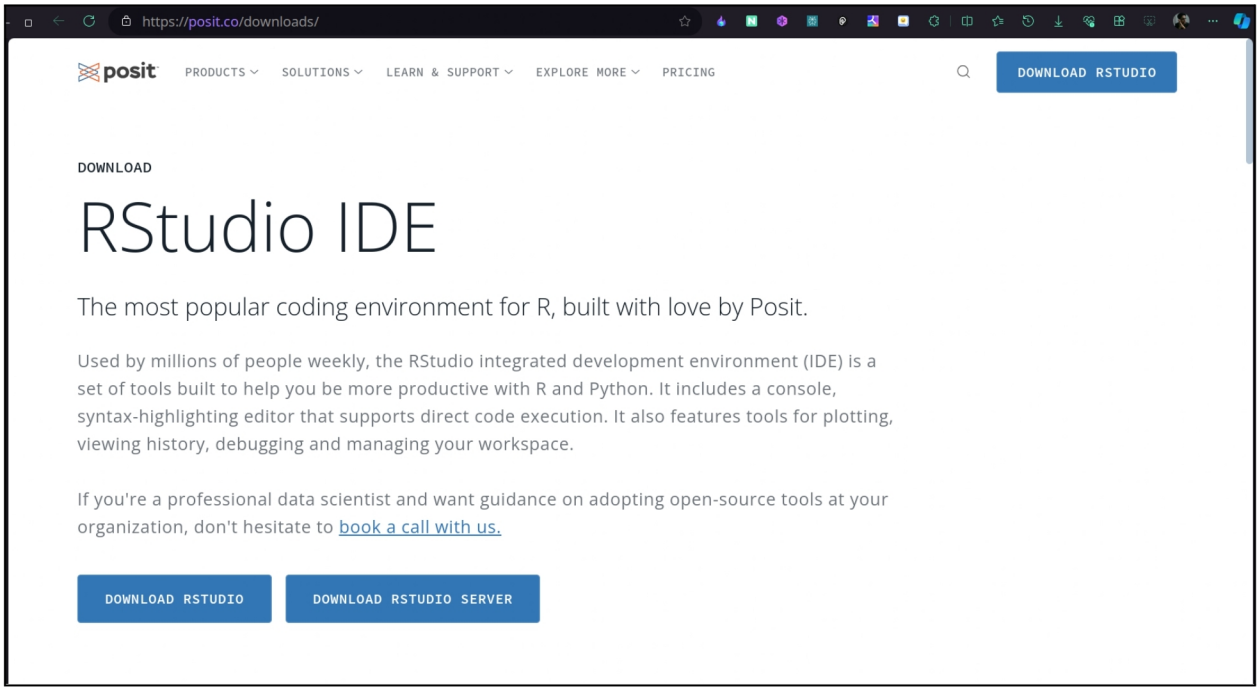
4. Setup R on your computer using default options and click proceed.



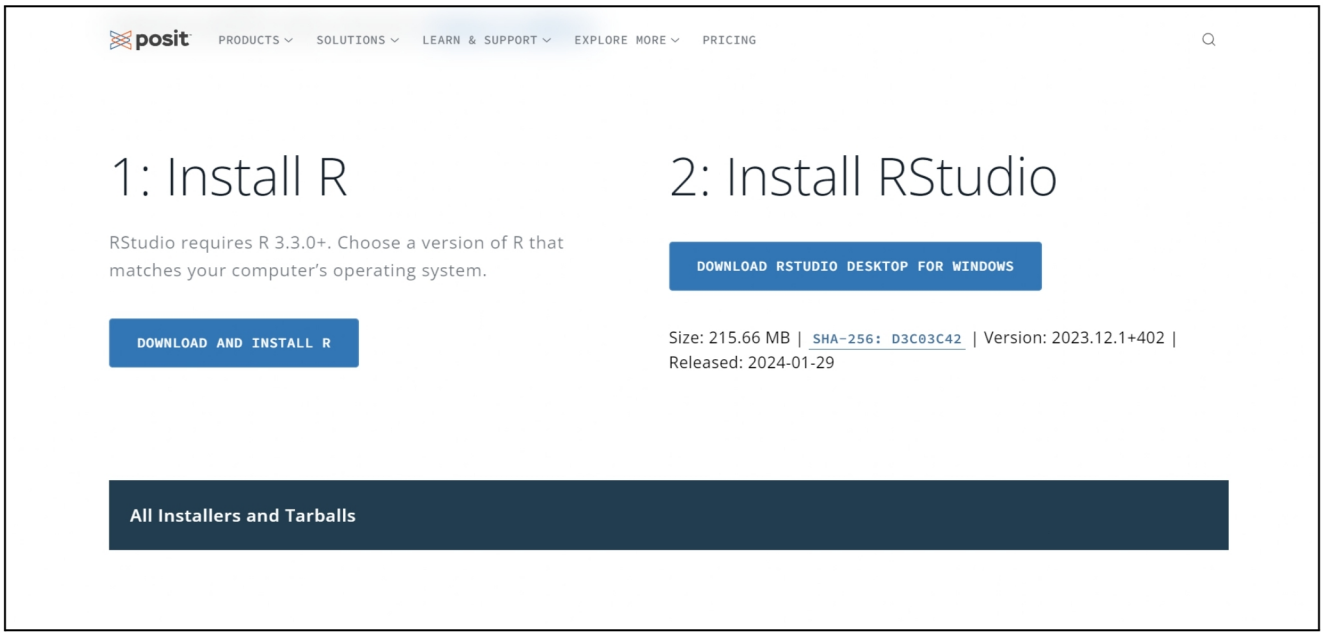
5. After the installation of R , its desktop entry shown on the Desktop.



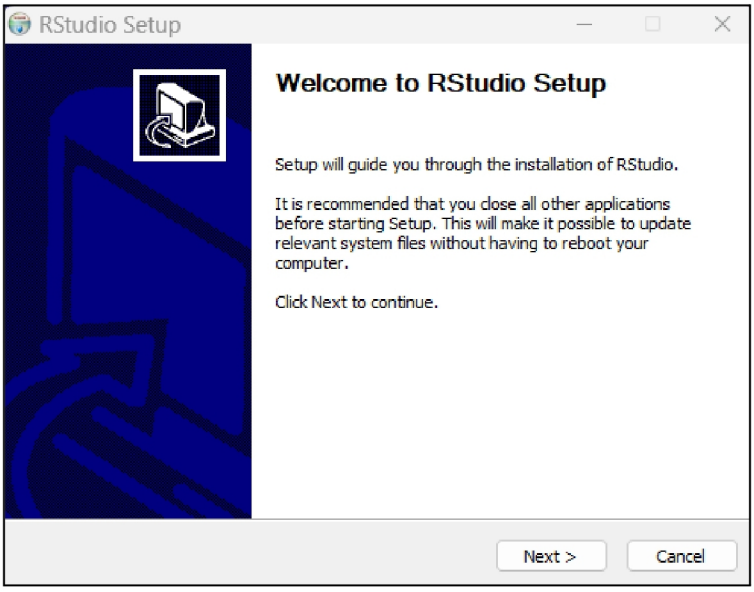
6. Next, download RStudio. Go to [Download RStudio - Posit](https://posit.co/downloads/) .



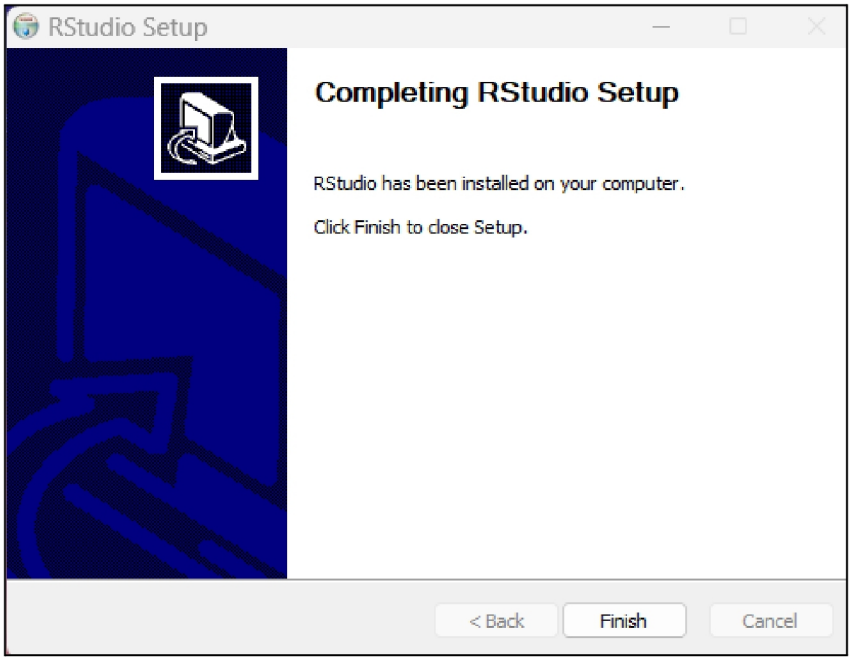
7. Click Download RStudio.



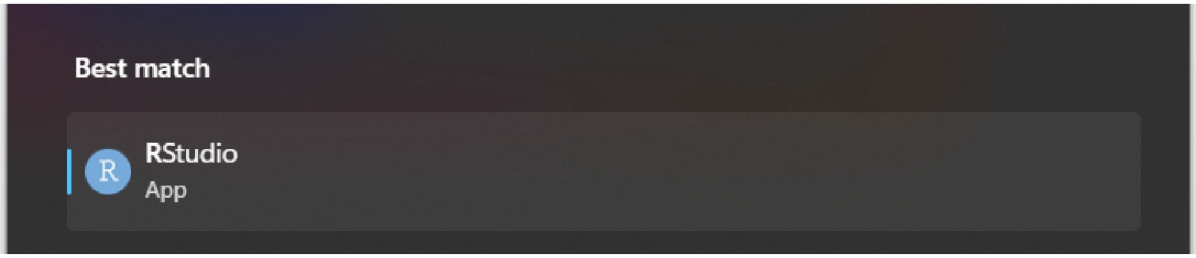
8. Once the packet has downloaded, the Welcome to RStudio Setup Wizard will open. Click Next and go through the installation steps.



9. Choose the default options and click finish.



10. After the Setup Wizard finishing the installation, RStudio can be launched from the Search Menu.

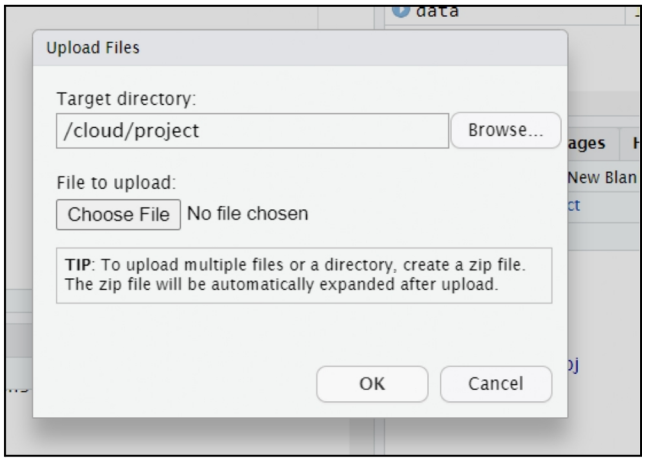


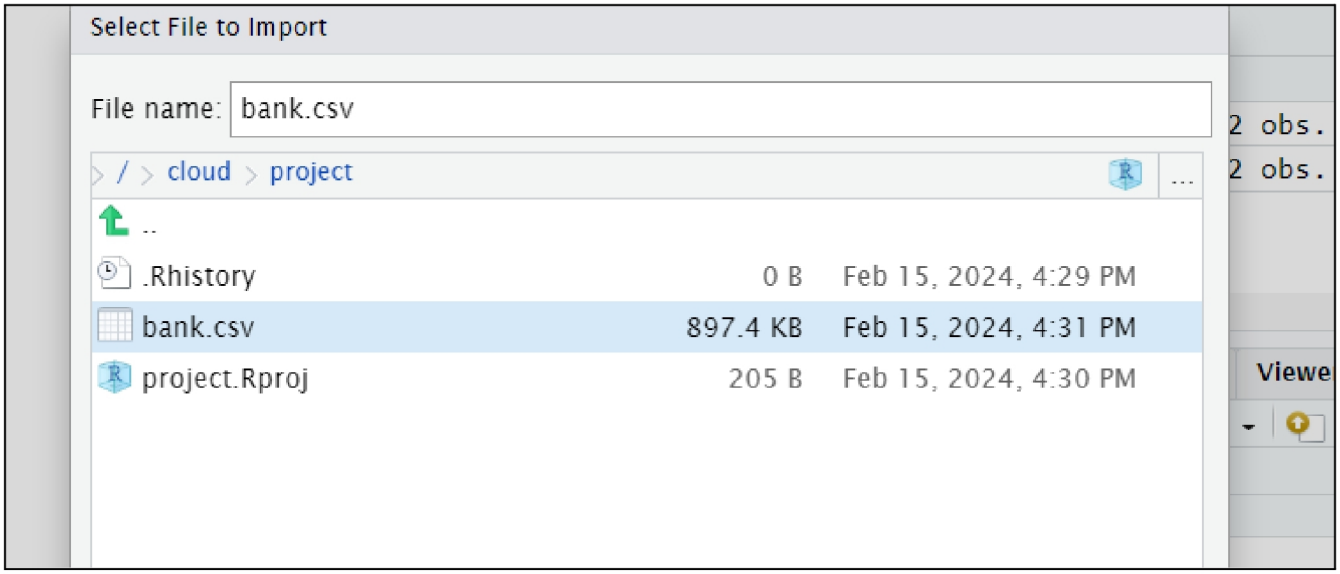
Analysis & Visualization of Bank Dataset-

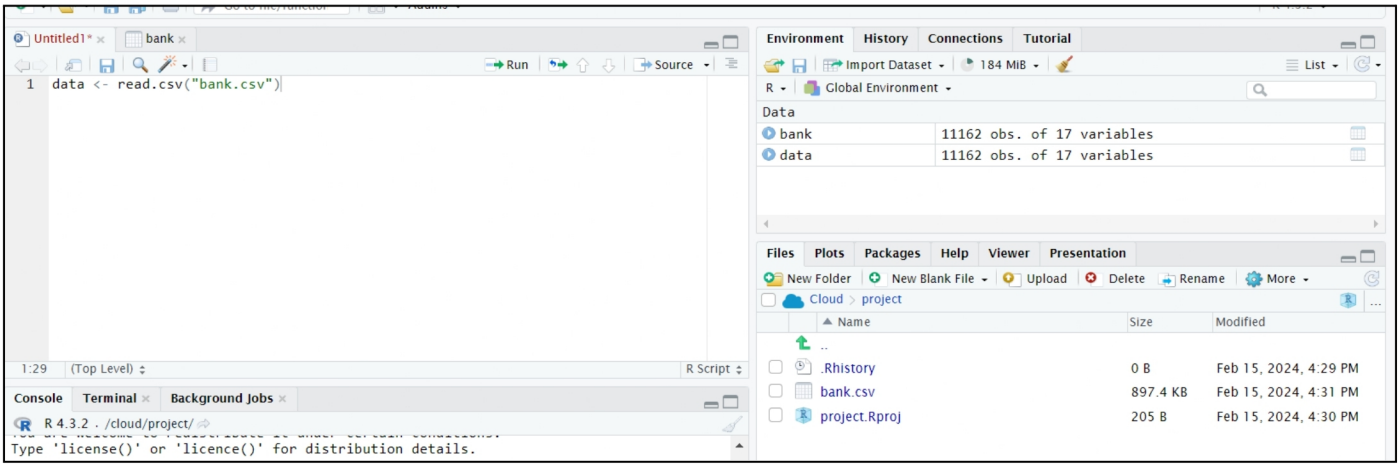
11. Go to [Kaggle - Bank Marketing Dataset] and download the Bank Dataset.

12. Extract archive and the bank.csv will be extracted.

13. Open Rstudio and Import the bank.csv dataset.







14. Insert the following code in the RStudio Console line by line.

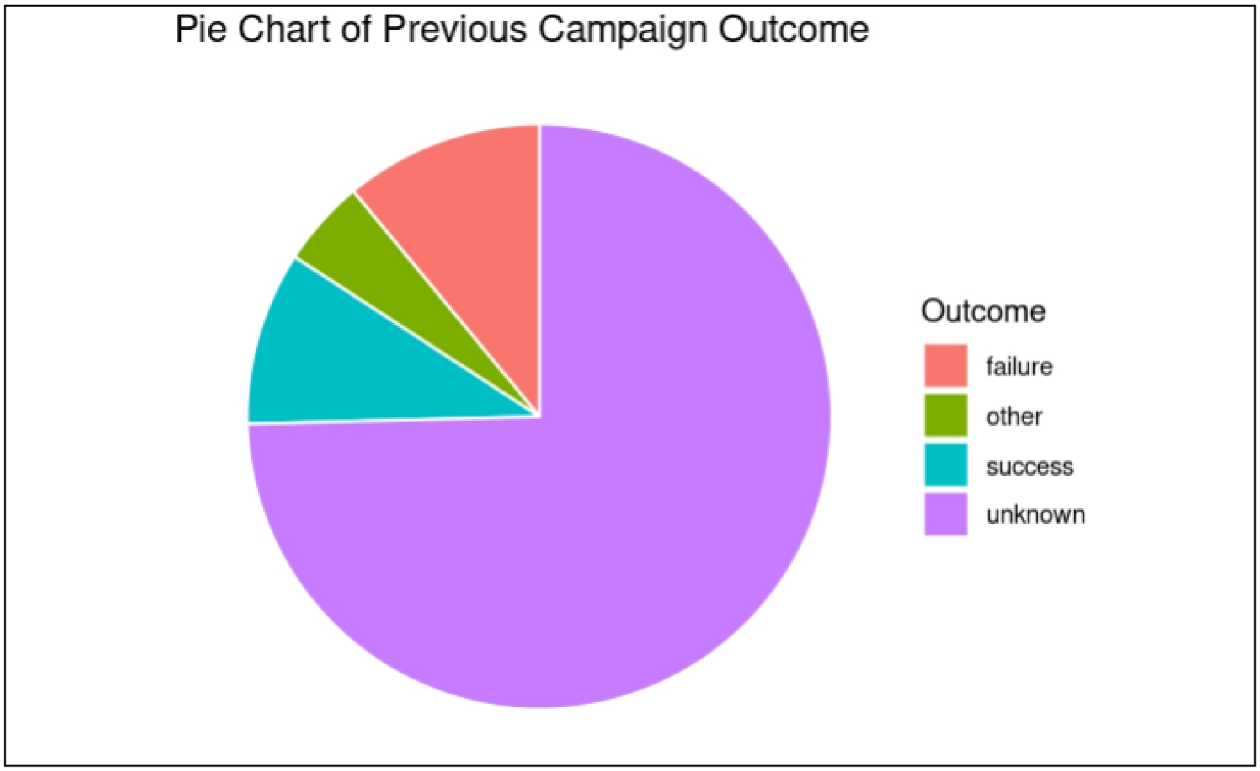
Code:

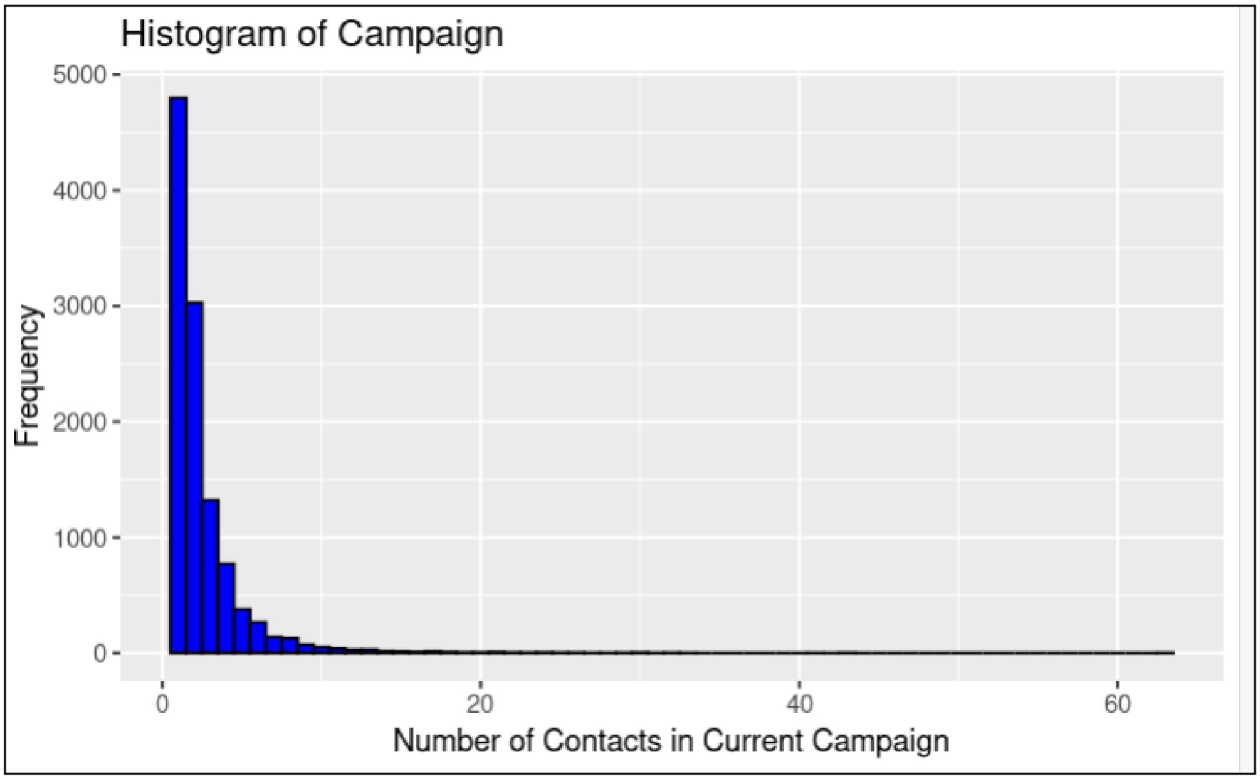
```
data <- read.csv("bank.csv")

#Visualization
install.packages("ggplot2")
# Create a pie chart for the 'poutcome' variable
ggplot(data, aes(x = "", fill = poutcome)) +
  geom_bar(width = 1, color = "white") +
  coord_polar("y", start = 0) +
  labs(title = "Pie Chart of Previous Campaign Outcome",
       fill = "Outcome",
       x = NULL,
       y = NULL) +
  theme_void() +
  theme(legend.position = "right")

#Histogram
# Create a histogram of the 'campaign' variable
ggplot(data, aes(x = campaign)) +
  geom_histogram(binwidth = 1, fill = "blue", color = "black") +
  labs(title = "Histogram of Campaign",
       x = "Number of Contacts in Current Campaign",
       y = "Frequency")



library(ggplot2)
library(dplyr)
# Bin the 'pdays' variable into categories
data <- mutate(data, pdays_category = cut(pdays, breaks = c(-1, 0, 100, 200, 300, 400, Inf)))

# Create a horizontal bar chart for the 'pdays' variable
ggplot(data, aes(x = pdays_category)) +
```

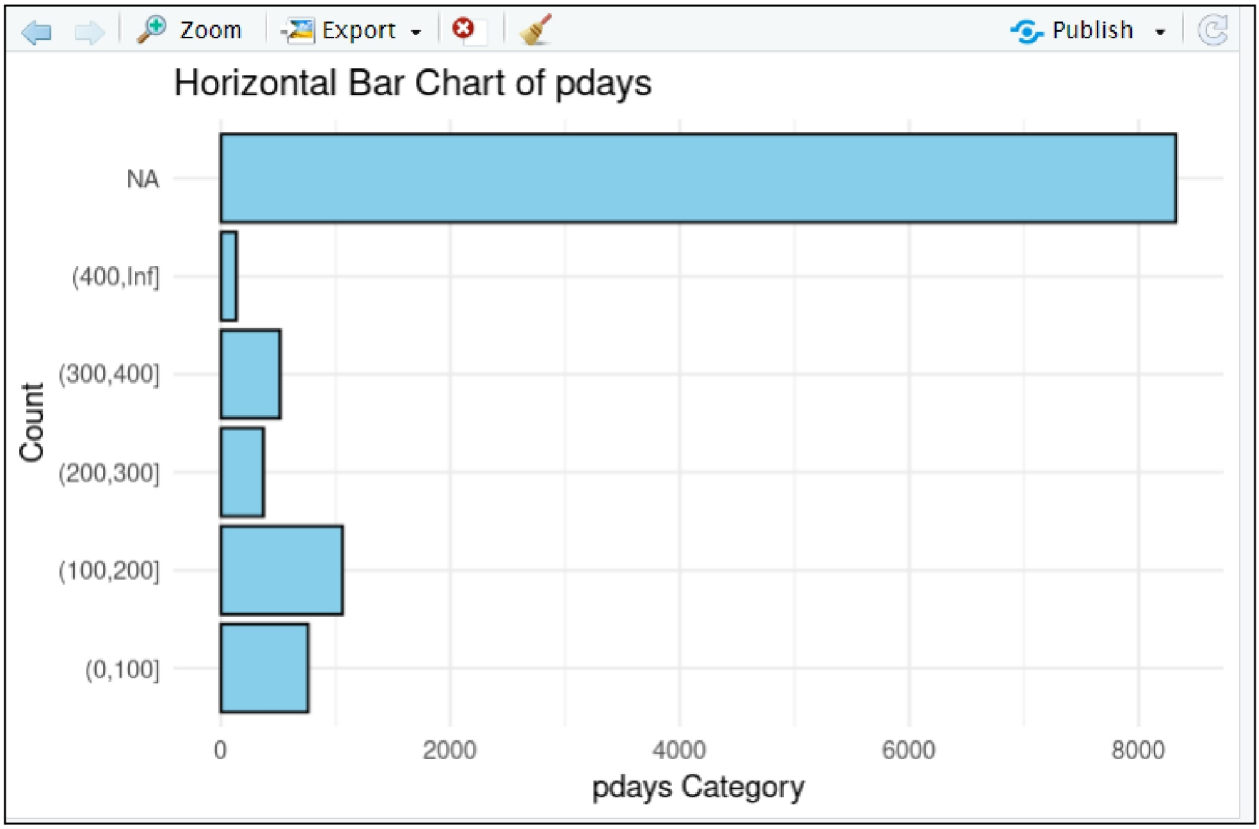
```
  geom_bar(fill = "skyblue", color = "black") +
  coord_flip() +  # This flips the axes to make it horizontal
  labs(title = "Horizontal Bar Chart of pdays",
       x = "Count",
       y = "pdays Category") +
  theme_minimal()


# Create a pie chart for the 'education' variable
install.packages("dplyr")
ggplot(data, aes(x = "", fill = education)) +
  geom_bar(width = 1, color = "white") +
  coord_polar("y", start = 0) +
  labs(title = "Pie Chart of Education Level",
       fill = "Education Level",
       x = NULL,
       y = NULL) +
  theme_void() +
  theme(legend.position = "right")

#Analysis
head(data)   # Display first few rows of the data
str(data)   # Get information about the data
summary(data)  # statistics for each variable
names(data) # Get the names of the attributes
```

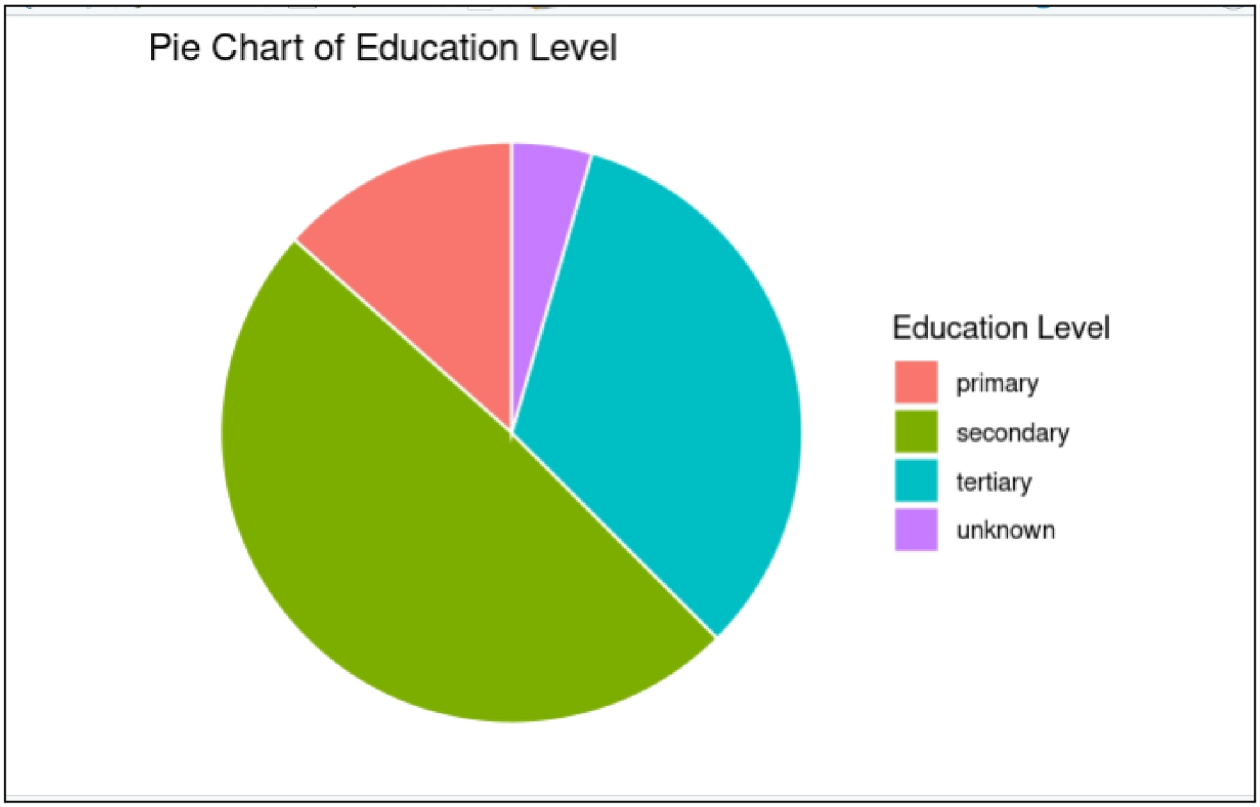
Output:

Visualization -




Histogram of Campaign
Frequency
5000
4000
3000
2000
1000
0
0
20
40
60
Number of Contacts in Current Campaign


Zoom
Export
Publish
Horizontal Bar Chart of pdays
Count
NA
(400,Inf]
(300,400]
(200,300]
(100,200]
(0,100]
0
2000
4000
6000
8000
pdays Category



Analysis-

```
> head(data)   # Show the first few rows of the data
  age        job marital education default balance housing loan contact day month
1  59     admin. married secondary      no    2343     yes   no unknown   5   may
2  56     admin. married secondary      no      45      no   no unknown   5   may
3  41 technician married secondary      no    1270     yes   no unknown   5   may
4  55   services married secondary      no    2476     yes   no unknown   5   may
5  54     admin. married  tertiary      no     184      no   no unknown   5   may
6  42 management  single  tertiary      no       0     yes  yes unknown   5   may
  duration campaign pdays previous poutcome deposit  balance_bins pdays_category
1     1042        1    -1        0  unknown     yes (2e+03,3e+03]           <NA>
2     1467        1    -1        0  unknown     yes     (0,1e+03]           <NA>
3     1389        1    -1        0  unknown     yes (1e+03,2e+03]           <NA>
4      579        1    -1        0  unknown     yes (2e+03,3e+03]           <NA>
5      673        2    -1        0  unknown     yes     (0,1e+03]           <NA>
```

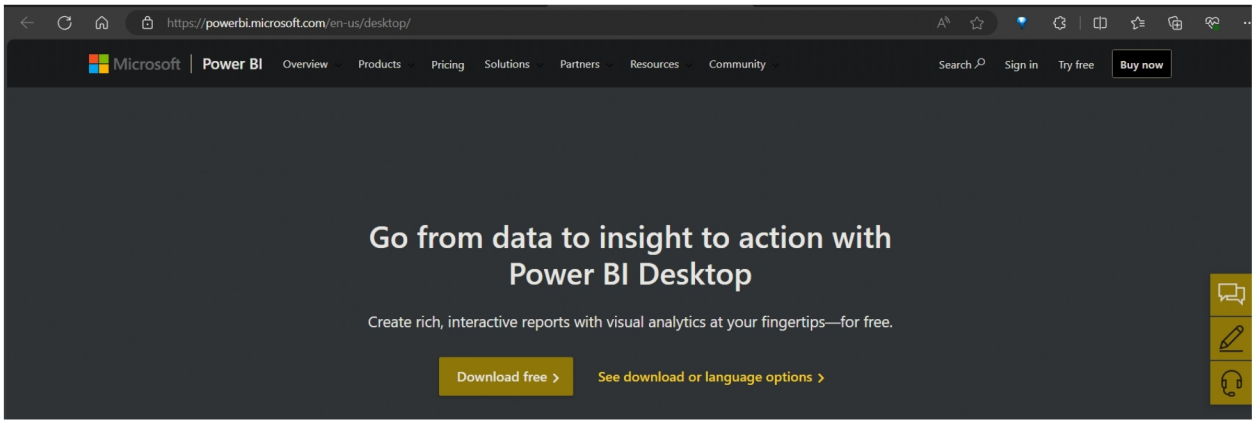
```
> str(data)   # Get information about the data
'data.frame':   11162 obs. of  19 variables:
 $ age           : int  59 56 41 55 54 42 56 60 37 28 ...
 $ job           : chr  "admin." "admin." "technician" "services" ...
 $ marital       : chr  "married" "married" "married" "married" ...
 $ education     : chr  "secondary" "secondary" "secondary" "secondary" ...
 $ default       : chr  "no" "no" "no" "no" ...
 $ balance       : int  2343 45 1270 2476 184 0 830 545 1 5090 ...
 $ housing       : chr  "yes" "no" "yes" "yes" ...
 $ loan          : chr  "no" "no" "no" "no" ...
 $ contact       : chr  "unknown" "unknown" "unknown" "unknown" ...
 $ day           : int  5 5 5 5 5 5 6 6 6 6 ...
```

```
> summary(data)  # Descriptive statistics for each variable
      age            job              marital           education
 Min.   :18.00   Length:11162       Length:11162       Length:11162
 1st Qu.:32.00   Class :character   Class :character   Class :character
 Median :39.00   Mode  :character   Mode  :character   Mode  :character
 Mean   :41.23
 3rd Qu.:49.00
 Max.   :95.00

   default             balance        housing             loan
 Length:11162       Min.   :-6847   Length:11162       Length:11162
 Class :character   1st Qu.:  122   Class :character   Class :character
 Mode  :character   Median :  550   Mode  :character   Mode  :character
                    Mean   : 1529
                    3rd Qu.: 1708
                    Max.   :81204
```

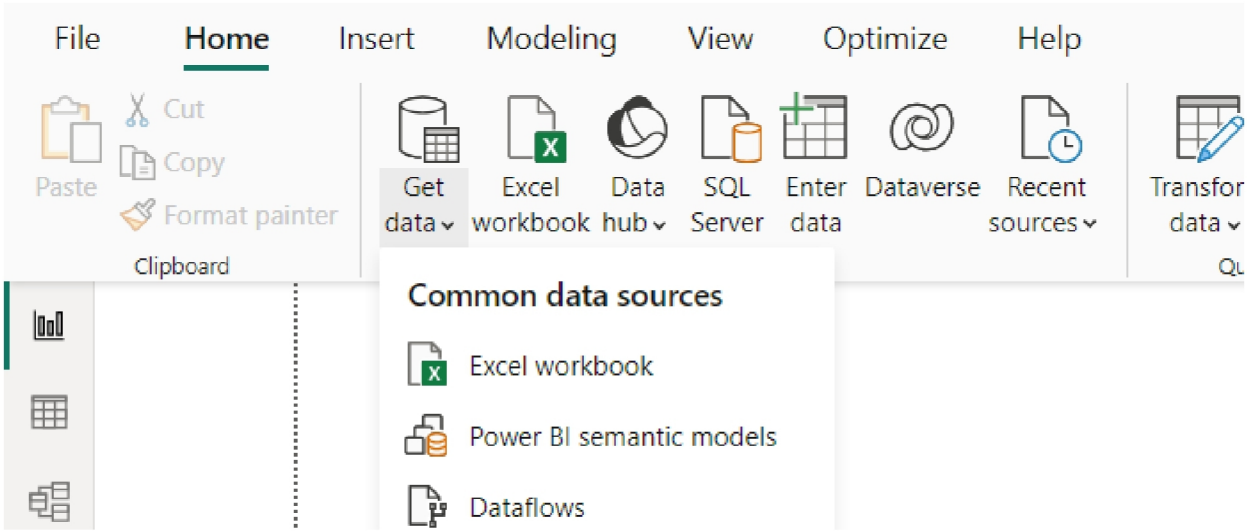
```
> names(data) # Get the names of the attributes
 [1] "age"            "job"            "marital"        "education"
 [5] "default"        "balance"        "housing"        "loan"
 [9] "contact"        "day"            "month"          "duration"
[13] "campaign"       "pdays"          "previous"       "poutcome"
[17] "deposit"        "balance_bins"   "pdays_category"
```

Task 3: Data Visualization using PowerBI

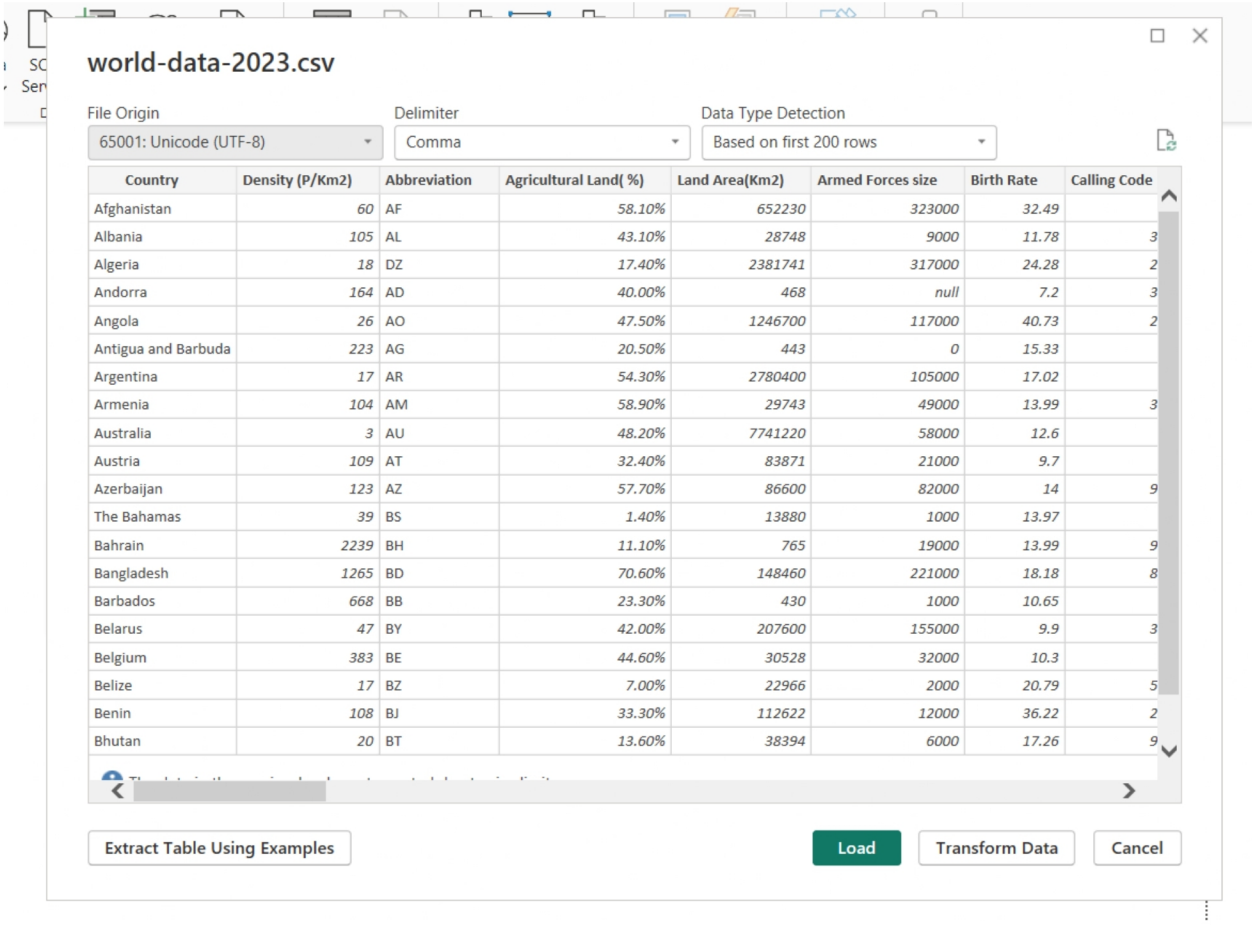
1. Install Powerbi by the link provided - https://powerbi.microsoft.com/en-us/desktop/



2. Install the required dataset and import it in Powerbi

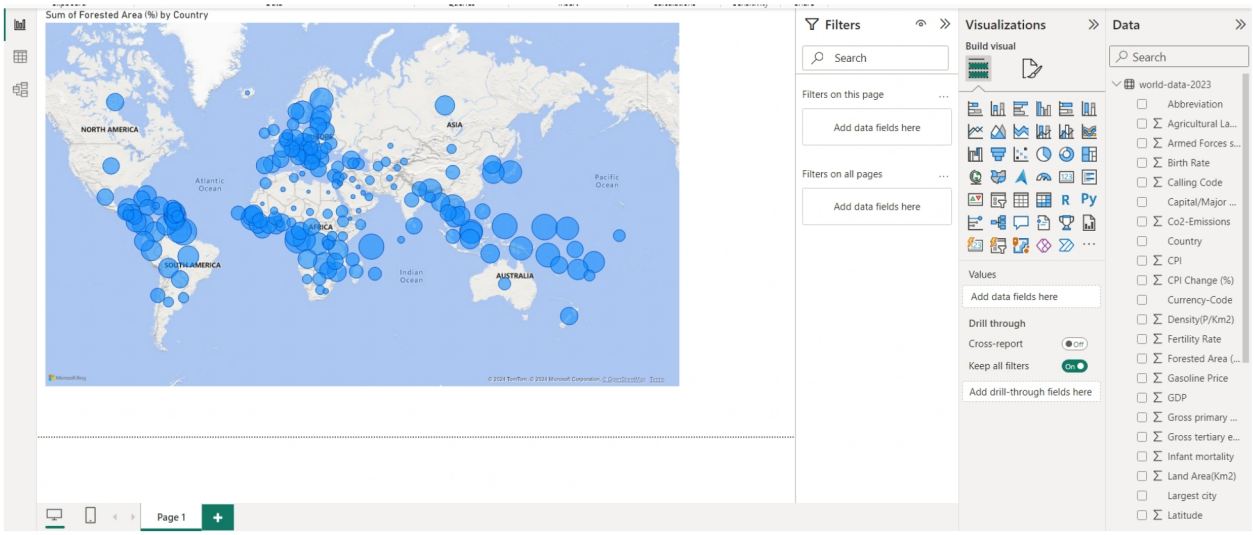


3. Click on Load Dataset.

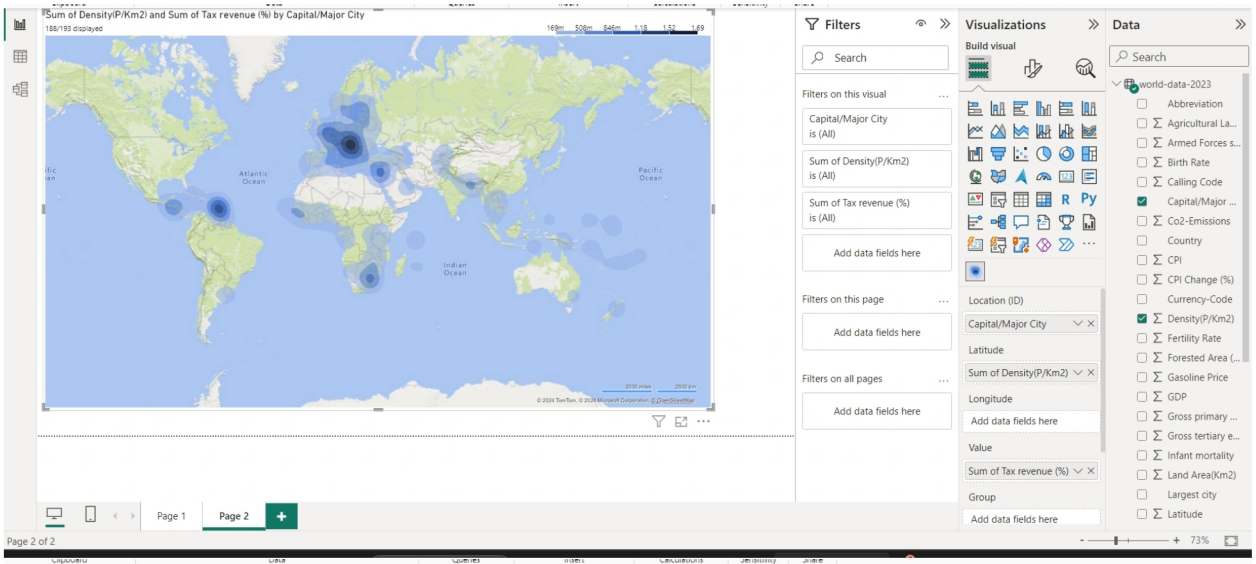


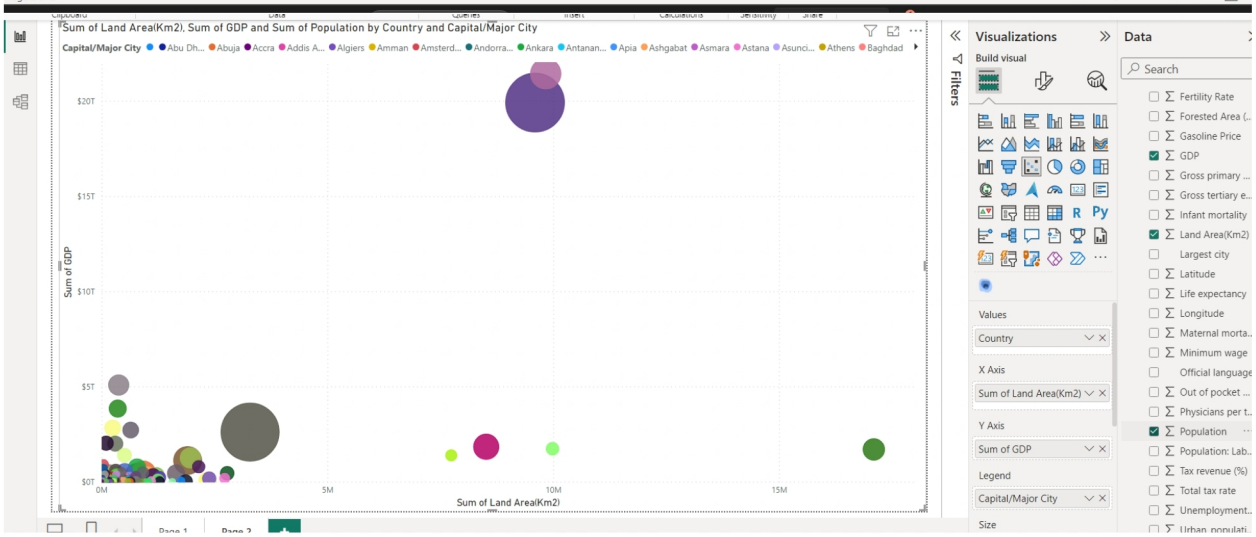
Country	Density (P/Km2)	Abbreviation	Agricultural Land(%)	Land Area(Km2)	Armed Forces size	Birth Rate	Calling Code
Afghanistan	60	AF	58.10%	652230	323000	32.49	
Albania	105	AL	43.10%	28748	9000	11.78	3
Algeria	18	DZ	17.40%	2381741	317000	24.28	2
Andorra	164	AD	40.00%	468	null	7.2	3
Angola	26	AO	47.50%	1246700	117000	40.73	2
Antigua and Barbuda	223	AG	20.50%	443	0	15.33	
Argentina	17	AR	54.30%	2780400	105000	17.02	
Armenia	104	AM	58.90%	29743	49000	13.99	3
Australia	3	AU	48.20%	7741220	58000	12.6	
Austria	109	AT	32.40%	83871	21000	9.7	
Azerbaijan	123	AZ	57.70%	86600	82000	14	9
The Bahamas	39	BS	1.40%	13880	1000	13.97	
Bahrain	2239	BH	11.10%	765	19000	13.99	9
Bangladesh	1265	BD	70.60%	148460	221000	18.18	8
Barbados	668	BB	23.30%	430	1000	10.65	
Belarus	47	BY	42.00%	207600	155000	9.9	3
Belgium	383	BE	44.60%	30528	32000	10.3	
Belize	17	BZ	7.00%	22966	2000	20.79	5
Benin	108	BJ	33.30%	112622	12000	36.22	2
Bhutan	20	BT	13.60%	38394	6000	17.26	9

3. Generate the map according to country density.

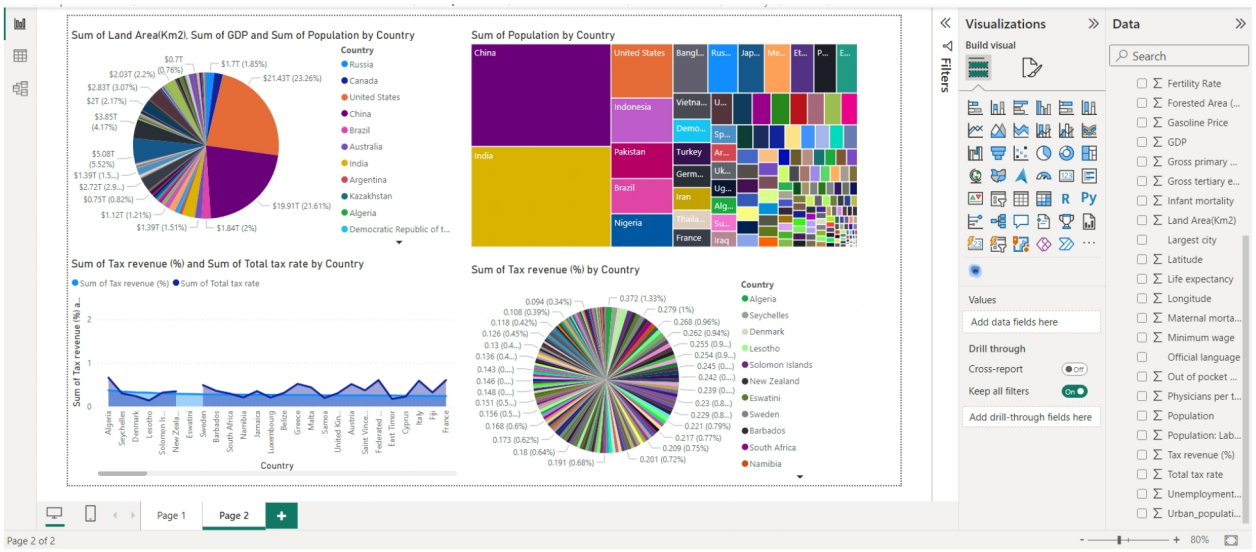


4. Generate the Heatmap according to country density/poulation or land area.





6. Do visualization of the dataset by various graphs like pie chart, line graph, etc.



TASK-4 CREATE and manipulate database in SPARK

Steps:

1. Install Spark in Google Collaboratory.

```
!pip install gspread-pandas
!pip install pyspark
```

```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.5.1
Master
local
AppName
Colab

2. Use RDD & Dataframes.

```
# Import necessary libraries
import gspread
from google.auth import default
from pyspark.sql.types import StructType, StructField, StringType

from google.colab import auth
auth.authenticate_user()

# from google.colab import drive
# drive.mount('/content/drive')

# Get Google Sheets credentials
creds, _ = default()

# Authorize gspread
gc = gspread.authorize(creds)

# Open the Google Sheets document by key
sheets_key = '1EuNWRKUOj0waKIgxf_hwstrFQezmRpzgtIYpkNnJj5M'
worksheet = gc.open_by_key(sheets_key).sheet1  # Adjust the sheet index as needed

# Get all values from the worksheet
data = worksheet.get_all_values()

# Define the Spark session
spark = SparkSession.builder.appName("GoogleSheetsToPySpark").getOrCreate()

# Define the schema based on the header
header = data[0]
schema = StructType([StructField(col, StringType(), True) for col in header])

# Create a PySpark DataFrame
rdd = spark.sparkContext.parallelize(data[1:])
df = spark.createDataFrame(rdd, schema)

# Show the PySpark DataFrame
df.show()
```

```
Mounted at /content/drive
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
|Serial No.|GRE Score|TOEFL Score|University Rating|SOP|LOR|CGPA|Research|Chance of Admit|
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
|         1|      337|        118|                4|4.5|4.5|9.65|       1|              1|
|         2|      324|        107|                4|  4|4.5|8.87|       1|           0.76|
|         3|      316|        104|                3|  3|3.5|   8|       1|           0.72|
|         4|      322|        110|                3|3.5|2.5|8.67|       1|            0.8|
|         5|      314|        103|                2|  2|  3|8.21|       0|           0.65|
|         6|      330|        115|                5|4.5|  3|9.34|       1|            0.9|
|         7|      321|        109|                3|  3|  4| 8.2|       1|           0.75|
|         8|      308|        101|                2|  3|  4| 7.9|       0|           0.68|
|         9|      302|        102|                1|  2|1.5|   8|       0|            0.5|
|        10|      323|        108|                3|3.5|  3| 8.6|       0|           0.45|
|        11|      325|        106|                3|3.5|  4| 8.4|       1|           0.52|
|        12|      327|        111|                4|  4|4.5|   9|       1|           0.84|
|        13|      328|        112|                4|  4|4.5| 9.1|       1|           0.78|
|        14|      307|        109|                3|  4|  3|   8|       1|           0.62|
|        15|      311|        104|                3|3.5|  2| 8.2|       1|           0.61|
|        16|      314|        105|                3|3.5|2.5| 8.3|       0|           0.54|
|        17|      317|        107|                3|  4|  3| 8.7|       0|           0.66|
|        18|      319|        106|                3|  4|  3|   8|       1|           0.65|
|        19|      318|        110|                3|  4|  3| 8.8|       0|           0.63|
|        20|      303|        102|                3|3.5|  3| 8.5|       0|           0.62|
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
only showing top 20 rows
```

3. Implement linear regression using sparkMlib.

```python
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline

# Select features and target column
feature_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']
target_column = 'Chance of Admit'

from pyspark.sql.functions import col
from pyspark.sql.types import FloatType

# Convert columns to appropriate numeric types
for col_name in feature_columns + [target_column]:
    df = df.withColumn(col_name, col(col_name).cast(FloatType()))

# Verify the schema after conversion
df.printSchema()


# Create a VectorAssembler to assemble the features into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Create a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol=target_column)

# Create a pipeline with the VectorAssembler and Linear Regression model
pipeline = Pipeline(stages=[assembler, lr])

# Fit the pipeline to the data
model = pipeline.fit(df)

# Make predictions
predictions = model.transform(df)

# Show the predictions
predictions.select("features", target_column, "prediction").show()

```

```
root
 |-- Serial No.: string (nullable = true)
 |-- GRE Score: float (nullable = true)
 |-- TOEFL Score: float (nullable = true)
 |-- University Rating: float (nullable = true)
 |-- SOP: float (nullable = true)
 |-- LOR: float (nullable = true)
 |-- CGPA: float (nullable = true)
 |-- Research: float (nullable = true)
 |-- Chance of Admit: float (nullable = true)

+--------------------+---------------+------------------+
|            features|Chance of Admit|        prediction|
+--------------------+---------------+------------------+
|[337.0,118.0,4.0,...|            1.0|0.9527419787810929|
|[324.0,107.0,4.0,...|           0.76|0.8042442709169288|
|[316.0,104.0,3.0,...|           0.72| 0.653362064831734|
|[322.0,110.0,3.0,...|            0.8| 0.744800849900342|
|[314.0,103.0,2.0,...|           0.65| 0.631671909170074|
|[330.0,115.0,5.0,...|            0.9|0.8748625451723717|
|[321.0,109.0,3.0,...|           0.75|0.7089061133853698|
|[308.0,101.0,2.0,...|           0.68|0.5965489750932171|
|[302.0,102.0,1.0,...|            0.5|0.5503679652106606|
|[323.0,108.0,3.0,...|           0.45|0.7169636074292443|
|[325.0,106.0,3.0,...|           0.52|0.7325297349799527|
|[327.0,111.0,4.0,...|           0.84| 0.836486918374747|
|[328.0,112.0,4.0,...|           0.78|0.8530258541953695|
|[307.0,109.0,3.0,...|           0.62|0.6436557091259056|
|[311.0,104.0,3.0,...|           0.61|0.6431819652204682|
|[314.0,105.0,3.0,...|           0.54|0.6476620556847543|
|[317.0,107.0,3.0,...|           0.66| 0.715595016570292|
|[319.0,106.0,3.0,...|           0.65|0.6577263403901585|
|[318.0,110.0,3.0,...|           0.63|0.7377331438983201|
|[303.0,102.0,3.0,...|           0.62|0.6508501135082854|
+--------------------+---------------+------------------+
only showing top 20 rows
```

```python
from pyspark.ml.evaluation import RegressionEvaluator

# Evaluate the model using Mean Squared Error (MSE)
evaluator = RegressionEvaluator(labelCol=target_column, predictionCol="prediction", metricName="mse")
mse = evaluator.evaluate(predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Evaluate the model using R-squared
evaluator = RegressionEvaluator(labelCol=target_column, predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)

# Convert R-squared to percentage
r2_percentage = r2 * 100

print(f"R-squared: {r2_percentage:.2f}%")
```

```
Mean Squared Error (MSE): 0.0035432110127672583
R-squared: 82.25%
```

Full Google Colab

TASK-4 CREATE and manipulate database in SPARK.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

RAM Disk Colab AI

Comment Share

[1]
```
!pip install gspread-pandas
!pip install pyspark
```

```
Collecting gspread-pandas
  Downloading gspread_pandas-3.3.0-py2.py3-none-any.whl (27 kB)
Collecting gspread<6,>=5.0.0 (from gspread-pandas)
  Downloading gspread-5.12.4-py3-none-any.whl (49 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 49.5/49.5 kB 2.8 MB/s eta 0:00:00
Requirement already satisfied: pandas>=0.20.0 in /usr/local/lib/python3.10/dist-packages (from gspread-pandas) (1.5.3)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from gspread-pandas) (4.4.2)
Requirement already satisfied: google-auth in /usr/local/lib/python3.10/dist-packages (from gspread-pandas) (2.27.0)
Requirement already satisfied: google-auth-oauthlib in /usr/local/lib/python3.10/dist-packages (from gspread-pandas) (1.2.0)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth->gspread-pandas) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth->gspread-pandas) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth->gspread-pandas) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib->gspread-pandas) (1.3.1)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->gspread-pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->gspread-pandas) (2023.4)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.20.0->gspread-pandas) (1.25.2)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth->gspread-pandas) (0.6.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.1->pandas>=0.20.0->gspread-pandas) (1.16.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib->gspread-pandas) (
Requirement already satisfied: requests>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib->gspread-pandas) (
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-o
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthlib->gsp
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthli
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.0.0->requests-oauthlib>=0.7.0->google-auth-oauthli
Installing collected packages: gspread, gspread-pandas
  Attempting uninstall: gspread
    Found existing installation: gspread 3.4.2
    Uninstalling gspread-3.4.2:
      Successfully uninstalled gspread-3.4.2
Successfully installed gspread-5.12.4 gspread-pandas-3.3.0
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 2.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... done
  Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488491 sha256=d7d03b4e495ac96c76a2e8d020d4d367baad794e4e937f33fad80b2e3fb579e8
  Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
```

[2]
```
from pyspark.sql import SparkSession

spark = SparkSession.builder\
        .master("local")\
        .appName("Colab")\
        .config('spark.ui.port', '4050')\
        .getOrCreate()
spark
```

SparkSession - in-memory

SparkContext

Spark UI

Version
v3.5.1
Master
local
AppName
Colab

[3]
```
!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
!unzip -o ngrok-stable-linux-amd64.zip
get_ipython().system_raw('./ngrok http 4050 &')
!curl -s http://localhost:4040/api/tunnels
```

```
--2024-04-03 09:12:03--  https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
Resolving bin.equinox.io (bin.equinox.io)... 52.202.168.65, 54.161.241.46, 54.237.133.81, ...
Connecting to bin.equinox.io (bin.equinox.io)|52.202.168.65|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13921656 (13M) [application/octet-stream]
Saving to: 'ngrok-stable-linux-amd64.zip'

ngrok-stable-linux- 100%[===================>]  13.28M  19.0MB/s    in 0.7s

2024-04-03 09:12:05 (19.0 MB/s) - 'ngrok-stable-linux-amd64.zip' saved [13921656/13921656]

Archive:  ngrok-stable-linux-amd64.zip
  inflating: ngrok
{"tunnels":[],"uri":"/api/tunnels"}
```

```
# Import necessary libraries
import gspread
from google.auth import default
from pyspark.sql.types import StructType, StructField, StringType

from google.colab import auth
auth.authenticate_user()

# from google.colab import drive
# drive.mount('/content/drive')

# Get Google Sheets credentials
creds, _ = default()

# Authorize gspread
gc = gspread.authorize(creds)

# Open the Google Sheets document by key
sheets_key = '1EuNWRKUOj0waKIgxf_hwstrFQezmRpzgtIYpkNnJj5M'
worksheet = gc.open_by_key(sheets_key).sheet1  # Adjust the sheet index as needed

# Get all values from the worksheet
data = worksheet.get_all_values()

# Define the Spark session
spark = SparkSession.builder.appName("GoogleSheetsToPySpark").getOrCreate()

# Define the schema based on the header
header = data[0]
schema = StructType([StructField(col, StringType(), True) for col in header])

# Create a PySpark DataFrame
rdd = spark.sparkContext.parallelize(data[1:])
df = spark.createDataFrame(rdd, schema)

# Show the PySpark DataFrame
df.show()
```

```
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
|Serial No.|GRE Score|TOEFL Score|University Rating|SOP|LOR|CGPA|Research|Chance of Admit|
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
|         1|      337|        118|                4|4.5|4.5|9.65|       1|              1|
|         2|      324|        107|                4|  4|4.5|8.87|       1|           0.76|
|         3|      316|        104|                3|  3|3.5|   8|       1|           0.72|
|         4|      322|        110|                3|3.5|2.5|8.67|       1|            0.8|
|         5|      314|        103|                2|  2|  3|8.21|       0|           0.65|
|         6|      330|        115|                5|4.5|  3|9.34|       1|            0.9|
|         7|      321|        109|                3|  3|  4| 8.2|       1|           0.75|
|         8|      308|        101|                2|  3|  4| 7.9|       0|           0.68|
|         9|      302|        102|                1|  2|1.5|   8|       0|            0.5|
|        10|      323|        108|                3|3.5|  3| 8.6|       0|           0.45|
|        11|      325|        106|                3|3.5|  4| 8.4|       1|           0.52|
|        12|      327|        111|                4|  4|4.5|   9|       1|           0.84|
|        13|      328|        112|                4|  4|4.5| 9.1|       1|           0.78|
|        14|      307|        109|                3|  4|  3|   8|       1|           0.62|
|        15|      311|        104|                3|3.5|  2| 8.2|       1|           0.61|
|        16|      314|        105|                3|3.5|2.5| 8.3|       0|           0.54|
|        17|      317|        107|                3|  4|  3| 8.7|       0|           0.66|
|        18|      319|        106|                3|  4|  3|   8|       1|           0.65|
|        19|      318|        110|                3|  4|  3| 8.8|       0|           0.63|
|        20|      303|        102|                3|3.5|  3| 8.5|       0|           0.62|
+----------+---------+-----------+-----------------+---+---+----+--------+---------------+
only showing top 20 rows
```

```
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
from pyspark.ml import Pipeline

# Select features and target column
feature_columns = ['GRE Score', 'TOEFL Score', 'University Rating', 'SOP', 'LOR', 'CGPA', 'Research']
target_column = 'Chance of Admit'

from pyspark.sql.functions import col
from pyspark.sql.types import FloatType

# Convert columns to appropriate numeric types
for col_name in feature_columns + [target_column]:
    df = df.withColumn(col_name, col(col_name).cast(FloatType()))

# Verify the schema after conversion
df.printSchema()


# Create a VectorAssembler to assemble the features into a single vector column
assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")

# Create a Linear Regression model
lr = LinearRegression(featuresCol="features", labelCol=target_column)

# Create a pipeline with the VectorAssembler and Linear Regression model
pipeline = Pipeline(stages=[assembler, lr])

# Fit the pipeline to the data
model = pipeline.fit(df)

# Make predictions
predictions = model.transform(df)

# Show the predictions
predictions.select("features", target_column, "prediction").show()

```

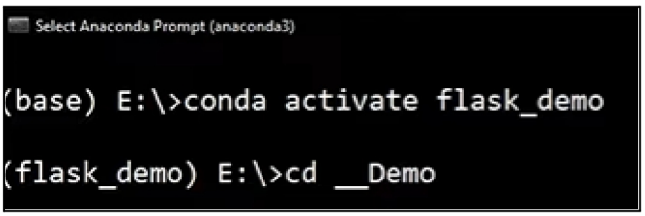
```
root
 |-- Serial No.: string (nullable = true)
 |-- GRE Score: float (nullable = true)
 |-- TOEFL Score: float (nullable = true)
 |-- University Rating: float (nullable = true)
 |-- SOP: float (nullable = true)
 |-- LOR: float (nullable = true)
 |-- CGPA: float (nullable = true)
 |-- Research: float (nullable = true)
 |-- Chance of Admit: float (nullable = true)

+--------------------+---------------+------------------+
|            features|Chance of Admit|        prediction|
+--------------------+---------------+------------------+
|[337.0,118.0,4.0,...|            1.0|0.9527419787810929|
|[324.0,107.0,4.0,...|           0.76|0.8042442709169288|
|[316.0,104.0,3.0,...|           0.72| 0.653362064831734|
|[322.0,110.0,3.0,...|            0.8| 0.744800849900342|
|[314.0,103.0,2.0,...|           0.65| 0.631671909170074|
|[330.0,115.0,5.0,...|            0.9|0.8748625451723717|
|[321.0,109.0,3.0,...|           0.75|0.7089061133853698|
|[308.0,101.0,2.0,...|           0.68|0.5965489750932171|
|[302.0,102.0,1.0,...|            0.5|0.5503679652106606|
|[323.0,108.0,3.0,...|           0.45|0.7169636074292443|
|[325.0,106.0,3.0,...|           0.52|0.7325297349799527|
|[327.0,111.0,4.0,...|           0.84| 0.836486918374747|
|[328.0,112.0,4.0,...|           0.78|0.8530258541953695|
|[307.0,109.0,3.0,...|           0.62|0.6436557091259056|
|[311.0,104.0,3.0,...|           0.61|0.6431819652204682|
|[314.0,105.0,3.0,...|           0.54|0.6476620556847543|
|[317.0,107.0,3.0,...|           0.66| 0.715595016570292|
|[319.0,106.0,3.0,...|           0.65|0.6577263403901585|
|[318.0,110.0,3.0,...|           0.63|0.7377331438983201|
|[303.0,102.0,3.0,...|           0.62|0.6508501135082854|
+--------------------+---------------+------------------+
only showing top 20 rows
```

```
from pyspark.ml.evaluation import RegressionEvaluator

# Evaluate the model using Mean Squared Error (MSE)
evaluator = RegressionEvaluator(labelCol=target_column, predictionCol="prediction", metricName="mse")
mse = evaluator.evaluate(predictions)
print(f"Mean Squared Error (MSE): {mse}")

# Evaluate the model using R-squared
evaluator = RegressionEvaluator(labelCol=target_column, predictionCol="prediction", metricName="r2")
r2 = evaluator.evaluate(predictions)

# Convert R-squared to percentage
r2_percentage = r2 * 100

print(f"R-squared: {r2_percentage:.2f}%")
```

```
Mean Squared Error (MSE): 0.0035432110127672583
R-squared: 82.25%
```

TASK-5 MLops using Heroku and Github actions - Submitted by Shobhit (2021UCS1618)

1. Make a new project - DEMO using ‘Anaconda’.

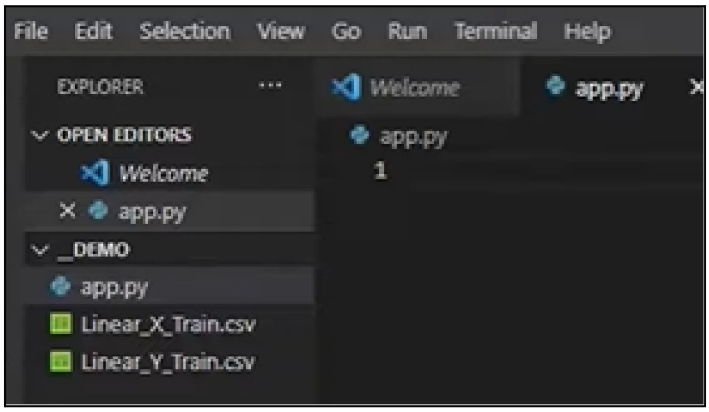


2. Install flask using ‘Anaconda pip’.

Dataset and Preprocessing

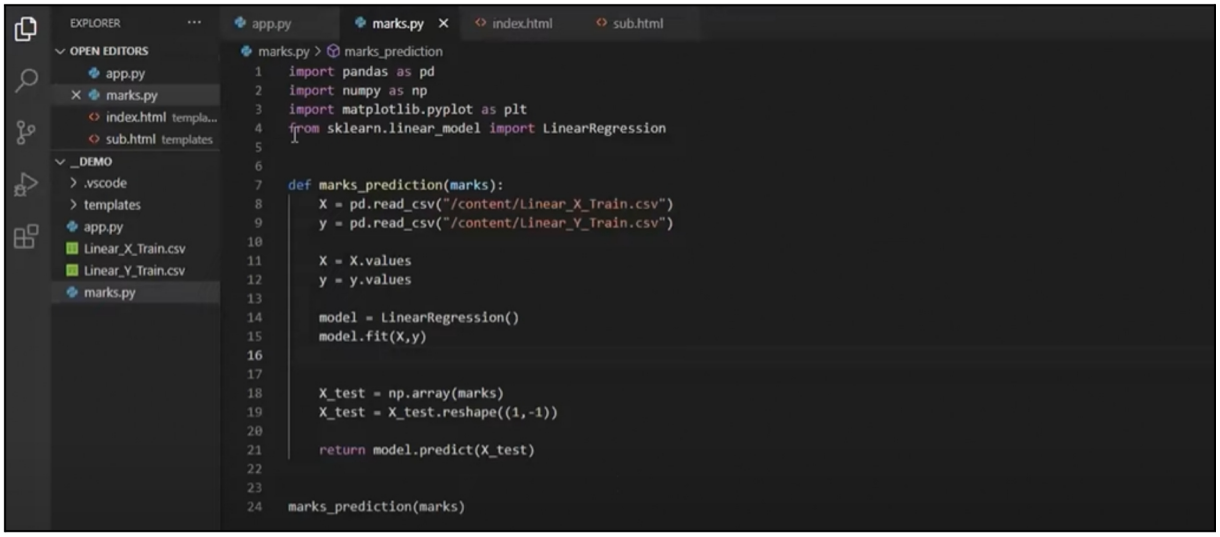
3. Import Class Score dataset (further used in linear regression) in this folder.

	A	B
1	Hours	Scores
2	2.5	21
3	5.1	47
4	3.2	27
5	8.5	75
6	3.5	30
7	1.5	20
8	9.2	88



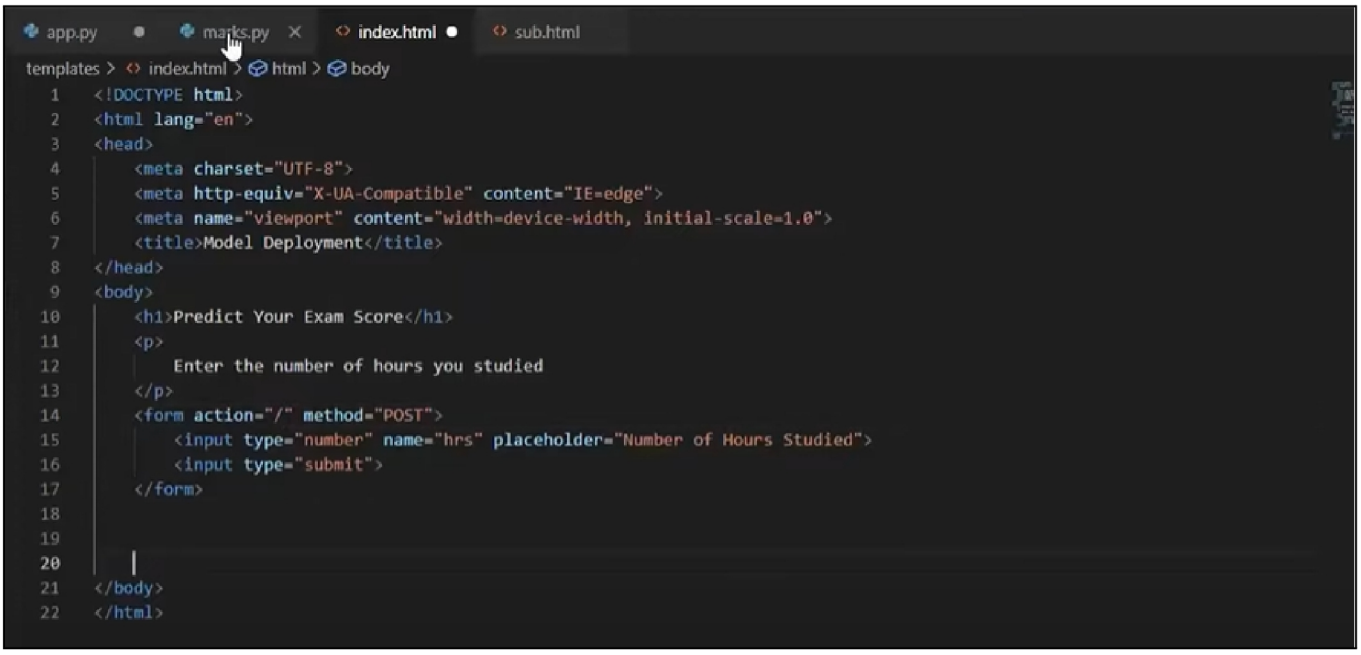
Encoding and Machine Learning Model

4. Create a new py file marks.py using that we will create and run the regression model.



Deployment

5. Create 3 files - index.html, sub.html & app.py.
   These files will be used to deploy our model.



Index.html

```python
from flask import Flask, render_template, request
import marks

app = Flask(__name__)

@app.route("/", methods=['POST'])
def hello():
    if request.method == "POST":
        hrs = request.form['hrs']
        marks_pred = marks.marks_prediction(hrs)
        print(marks_pred)

    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```

app.py

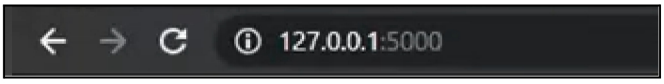
app.py index.html sub.html

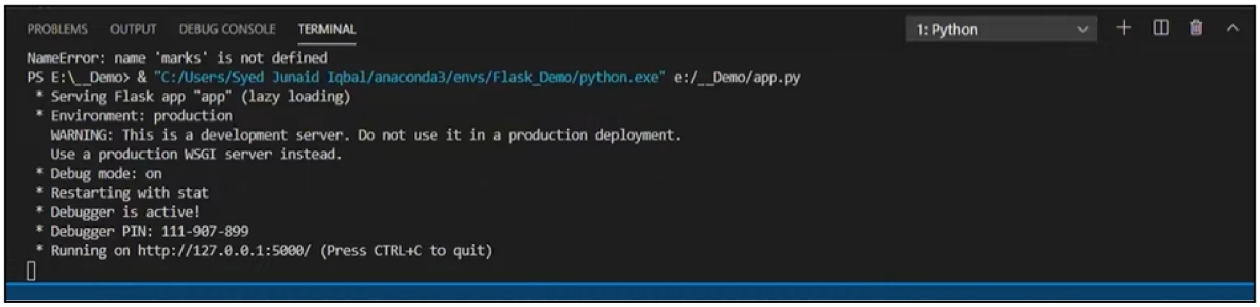
templates > sub.html > html > body > h4

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Submit page</title>
</head>
<body>

    <h4>{{ n }}</h4>

</body>
</html>
```

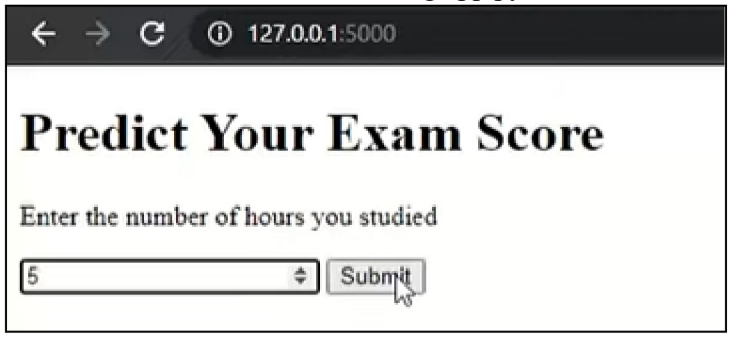
sub.html

6. On running app.py, our server will gets started on localhost:5000.





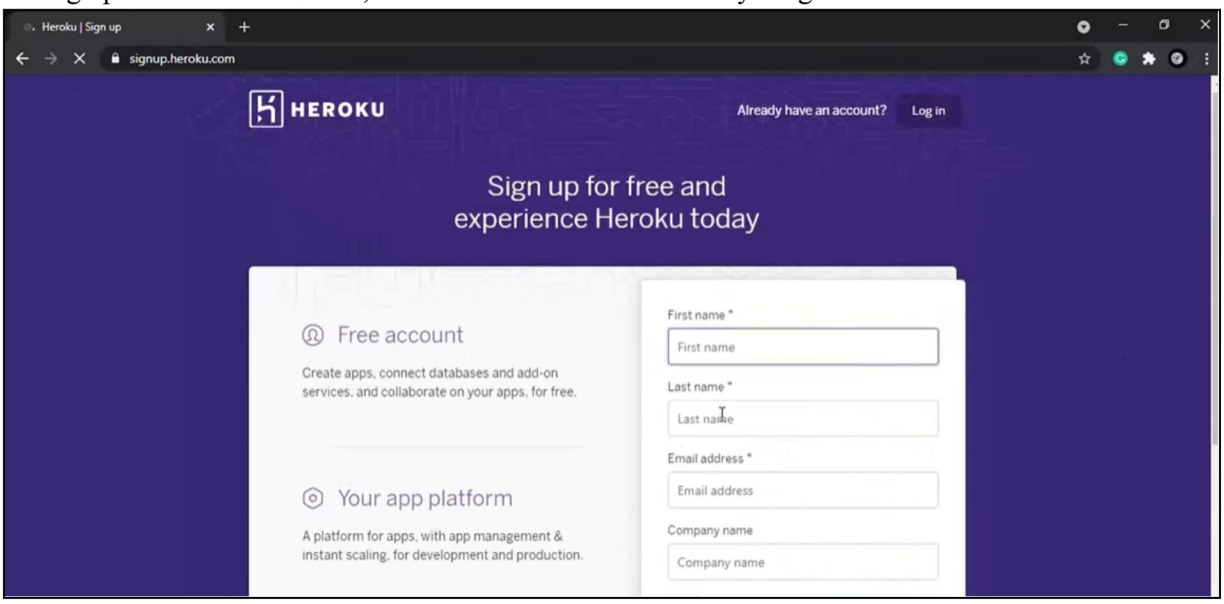
7. Go to localhost:5000 after re-running app.py and enter number of hours you studied.



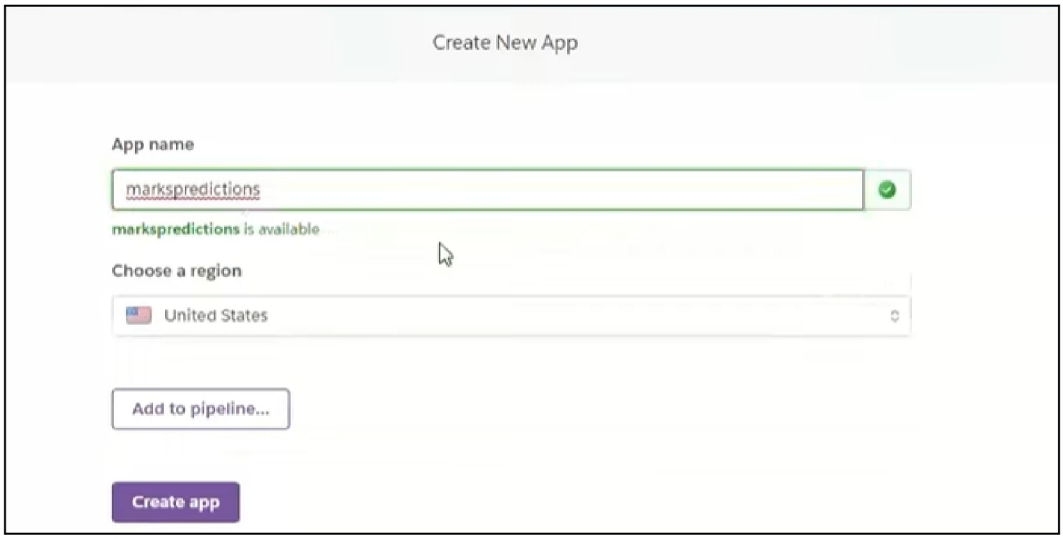
8. It will show predicted markes.

Deployment on Heroku

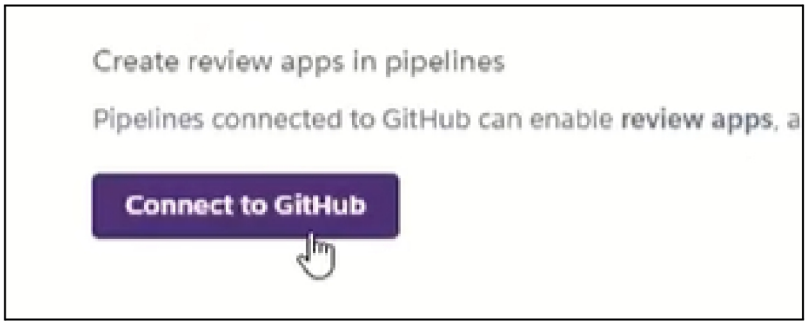
9. Setting up heroku - On Heroku, create new account & connect your github to it.



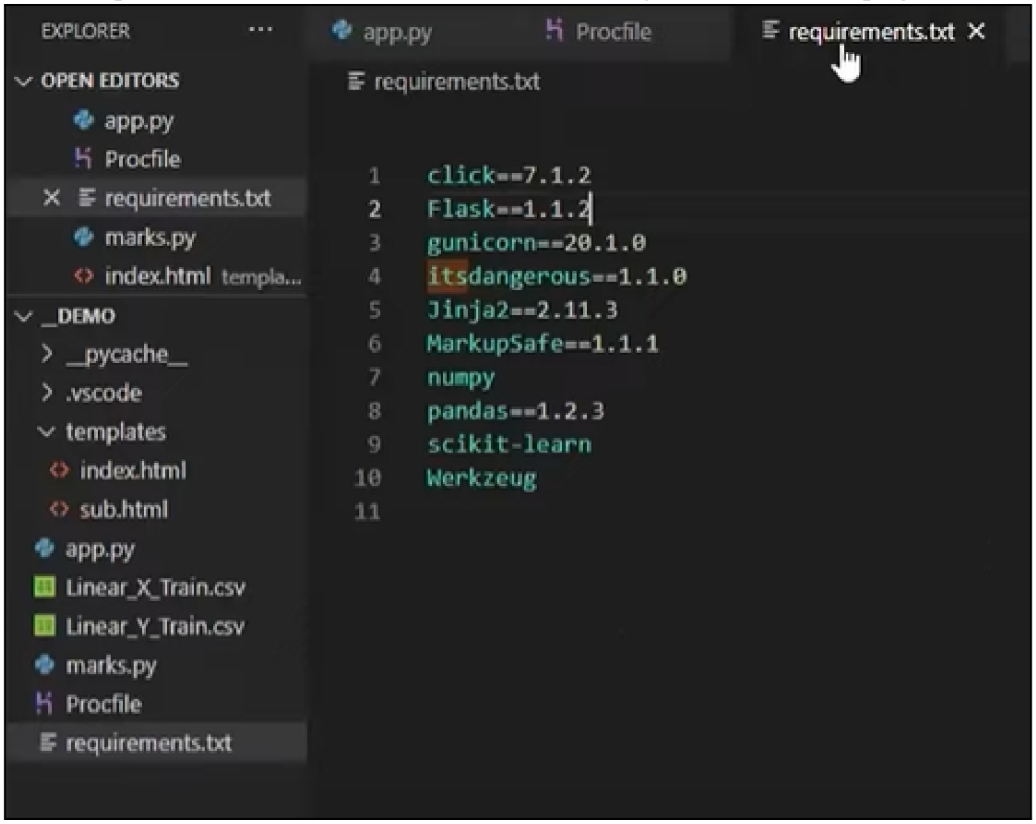
10. Create a new app on heroku



11. Connect to Github.

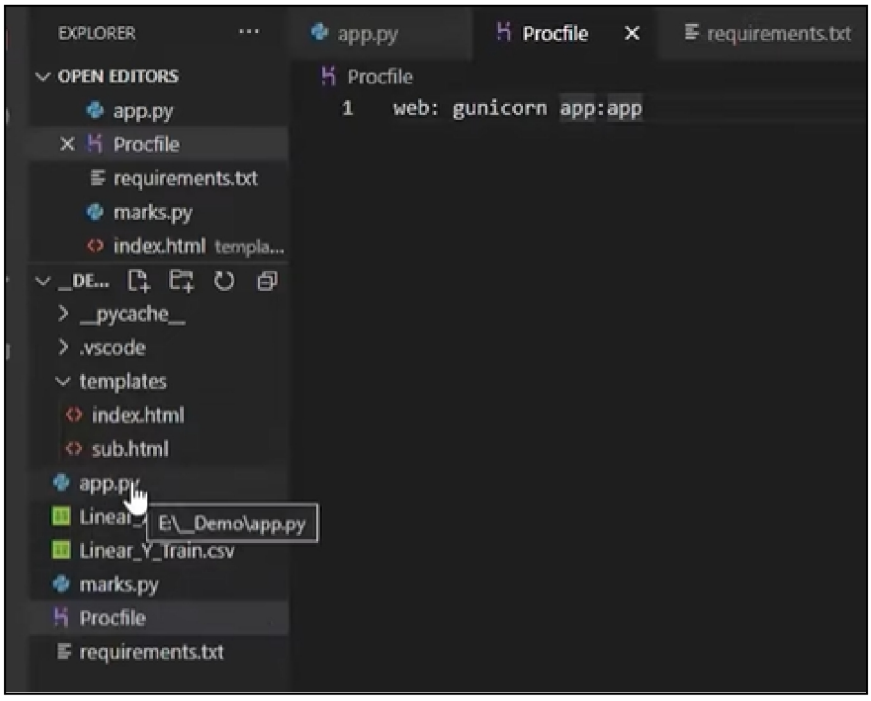


12. Create requirements.txt which will contain all necessary modules for this project.



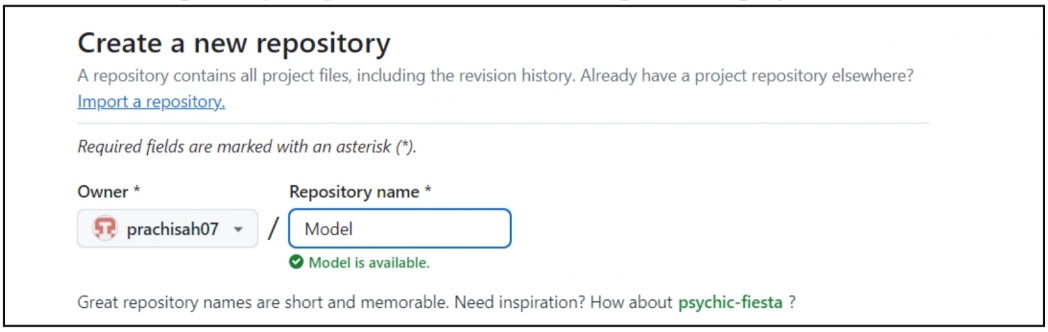
13. Install gunicorn in the environment and create Procfile.

```
(flask_demo) E:\__Demo>pip freeze > requirements.txt

(flask_demo) E:\__Demo>pip install gunicorn
Requirement already satisfied: gunicorn in c:\users\syed junaid iqbal\anaconda3\envs\flask_demo\lib
e-packages (20.1.0)
Requirement already satisfied: setuptools>=3.0 in c:\users\syed junaid iqbal\anaconda3\envs\flask_d
lib\site-packages (from gunicorn) (52.0.0.post20210125)
```

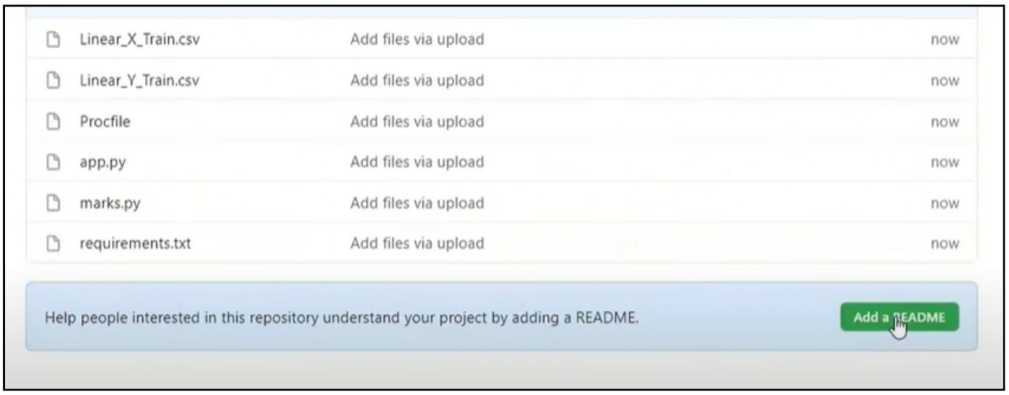


Incorporating GitHub actions

14. Create a new repository on github in which we will upload our project.

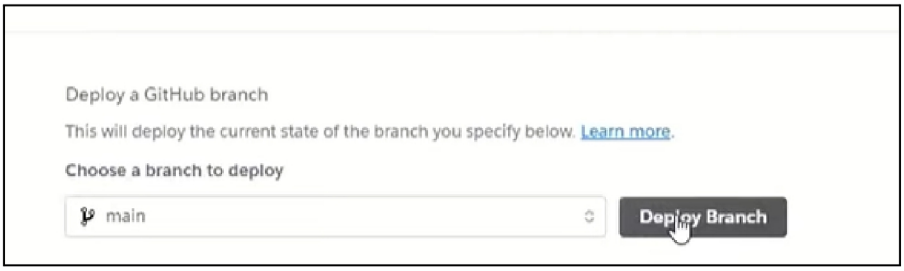


15. Commit the changes in the repository.



16. Search for your repository in github section of heroku and connect it.

17. Using github pages, we will deploy this app on heroku.



18. The app has been deployed on heroku and the link has been generated for the web page.