
# UNIVERSITY OF PETROLEUM AND ENERGY STUDIES
## School of Computer Science

## Applications of ML in industry Lab
## Lab File
## (2023-2024)

6th Semester

**Submitted To:**

Ms. Pallabi Sharma
*School of Computer Science*

**Submitted By:**

Aditi Tiwari
*B. Tech. CSE AIML*
*(non-hons.)*
*500090865*
*Batch 1*

# Index

# Experiment 1
# Introduction to Pandas and Numpy

---

## Introduction to pandas

### Task 1: Basic DataFrame Operations

I. Download a dataset of your choice (CSV, Excel, or any other format). And load the dataset into a Pandas DataFrame.

II. Display the first 5 rows of the dataset. Check for missing values and handle them appropriately. Get a summary of the dataset using describe().

III. Select a subset of columns from the DataFrame. Use both label-based and position-based indexing. Create a new DataFrame by filtering rows based on a condition.

[44]:
```
import pandas as pd
car_sales = pd.read_csv("car-sales-Copy1.csv")
car_sales.head(5)
```

[44]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| 0 | Toyota | White | 150043 | 4.0 | 4,000 |
| 1 | Honda | NaN | 87899 | 4.0 | 5,000 |
| 2 | Toyota | Blue | 32549 | 3.0 | 7,000 |
| 3 | BMW | Black | 11179 | 5.0 | 22,000 |
| 4 | Nissan | White | 213095 | 4.0 | 3,500 |

[45]:
```
car_sales[car_sales['Make'] == "Toyota"]
```

[45]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| 0 | Toyota | White | 150043 | 4.0 | 4,000 |
| 2 | Toyota | Blue | 32549 | 3.0 | 7,000 |
| 5 | Toyota | Green | 99213 | NaN | 4,500 |
| 8 | Toyota | White | 60000 | NaN | 6,250 |

### Task 2: Data Cleaning and Preprocessing

I. Identify missing values in the dataset. Decide on a strategy to handle missing values (e.g., imputation or removal). Implement the chosen strategy and explain the reasoning.

II. Create a new column by applying a mathematical operation on existing columns. Convert a categorical variable into numerical representation (e.g., using one-hot encoding).

III. Group the data by a specific column. Apply aggregation functions (sum, mean, count) to the grouped data. Present the results in a meaningful way.

[46]:
```
#Remove all rows wit NULL values from the DataFrame.
car_sales_missing_dropped = car_sales.dropna()
car_sales_missing_dropped
```

[46]:

| | Make | Colour | Odometer (KM) | Doors | Price |
|---|---|---|---|---|---|
| 0 | Toyota | White | 150043 | 4.0 | 4,000 |
| 2 | Toyota | Blue | 32549 | 3.0 | 7,000 |
| 3 | BMW | Black | 11179 | 5.0 | 22,000 |
| 4 | Nissan | White | 213095 | 4.0 | 3,500 |
| 7 | Honda | Blue | 54738 | 4.0 | 7,000 |
| 9 | Nissan | White | 31600 | 4.0 | 9,700 |

```
fuel_economy = [7.5, 9.2, 8.3, 7.3, 8.2, 9.4, 8.6, 7.4, 8.8, 7.8]
car_sales['Fuel (per 100Km)'] = fuel_economy
car_sales
```

[47]:

| | Make | Colour | Odometer (KM) | Doors | Price | Fuel (per 100Km) |
|---|---|---|---|---|---|---|
| 0 | Toyota | White | 150043 | 4.0 | 4,000 | 7.5 |
| 1 | Honda | NaN | 87899 | 4.0 | 5,000 | 9.2 |
| 2 | Toyota | Blue | 32549 | 3.0 | 7,000 | 8.3 |
| 3 | BMW | Black | 11179 | 5.0 | 22,000 | 7.3 |
| 4 | Nissan | White | 213095 | 4.0 | 3,500 | 8.2 |
| 5 | Toyota | Green | 99213 | NaN | 4,500 | 9.4 |
| 6 | Honda | NaN | 45698 | 4.0 | 7,500 | 8.6 |
| 7 | Honda | Blue | 54738 | 4.0 | 7,000 | 7.4 |
| 8 | Toyota | White | 60000 | NaN | 6,250 | 8.8 |
| 9 | Nissan | White | 31600 | 4.0 | 9,700 | 7.8 |

[48]:
```
# to find fuel used in car's lifetime
car_sales['Total fuel used (L)'] = car_sales['Odometer (KM)']/100 * car_sales['Fuel (per 100Km)']
car_sales
```

[48]:

| | Make | Colour | Odometer (KM) | Doors | Price | Fuel (per 100Km) | Total fuel used (L) |
|---|---|---|---|---|---|---|---|
| **0** | Toyota | White | 150043 | 4.0 | 4,000 | 7.5 | 11253.225 |
| **1** | Honda | NaN | 87899 | 4.0 | 5,000 | 9.2 | 8086.708 |
| **2** | Toyota | Blue | 32549 | 3.0 | 7,000 | 8.3 | 2701.567 |
| **3** | BMW | Black | 11179 | 5.0 | 22,000 | 7.3 | 816.067 |
| **4** | Nissan | White | 213095 | 4.0 | 3,500 | 8.2 | 17473.790 |
| **5** | Toyota | Green | 99213 | NaN | 4,500 | 9.4 | 9326.022 |
| **6** | Honda | NaN | 45698 | 4.0 | 7,500 | 8.6 | 3930.028 |
| **7** | Honda | Blue | 54738 | 4.0 | 7,000 | 7.4 | 4050.612 |
| **8** | Toyota | White | 60000 | NaN | 6,250 | 8.8 | 5280.000 |
| **9** | Nissan | White | 31600 | 4.0 | 9,700 | 7.8 | 2464.800 |

[49]:
```
# Remove commas from numeric columns
car_sales['Odometer (KM)'] = car_sales['Odometer (KM)'].replace({',': ''}, regex=True).astype(float)
car_sales['Price'] = car_sales['Price'].replace({',': ''}, regex=True).astype(float)

# Grouping by 'Make'
grouped_data = car_sales.groupby('Make')

# Applying aggregation functions
sum_data = grouped_data.agg({'Odometer (KM)': 'sum', 'Price': 'sum'})
mean_data = grouped_data.agg({'Odometer (KM)': 'mean', 'Price': 'mean'})
count_data = grouped_data.size().reset_index(name='count')

# Presenting the results
print("Total Odometer and Price by Make:", sum_data)
print("\nAverage Odometer and Price by Make:", mean_data)
print("\nNumber of Transactions by Make:", count_data)
```

```
Total Odometer and Price by Make:         Odometer (KM)    Price
Make
BMW            11179.0  22000.0
Honda         188335.0  19500.0
Nissan        244695.0  13200.0
Toyota        341805.0  21750.0

Average Odometer and Price by Make:          Odometer (KM)    Price
Make
BMW       11179.000000  22000.0
Honda     62778.333333   6500.0
Nissan   122347.500000   6600.0
Toyota    85451.250000   5437.5

Number of Transactions by Make:      Make  count
0     BMW      1
1   Honda      3
2  Nissan      2
3  Toyota      4
```

## Task 3: Loading and analyzing the dataset

Load two different datasets. Merge them using different types of joins (inner, outer, left, right). Analyze the impact of each type of join on the merged dataset.

[51]:
```
import pandas as pd

# Creating and saving two dataframes as CSV files
data1 = {'ID': [1, 2, 3], 'Name': ['Alice', 'Bob', 'Charlie']}
data2 = {'ID': [2, 3, 4], 'Age': [25, 30, 22]}

df1 = pd.DataFrame(data1)
df2 = pd.DataFrame(data2)

# Save dataframes to CSV files
df1.to_csv('data1.csv', index=False)
df2.to_csv('data2.csv', index=False)

# Load datasets
df1 = pd.read_csv('data1.csv')
df2 = pd.read_csv('data2.csv')
```

```
# Display the original datasets
print("DataFrame 1:")
print(df1)

print("\nDataFrame 2:")
print(df2)

# Inner Join
inner_merged = pd.merge(df1, df2, on='ID', how='inner')
print("\nInner Join Result:")
print(inner_merged)

# Outer Join
outer_merged = pd.merge(df1, df2, on='ID', how='outer')
print("\nOuter Join Result:")
print(outer_merged)

# Left Join
left_merged = pd.merge(df1, df2, on='ID', how='left')
print("\nLeft Join Result:")
print(left_merged)

# Right Join
right_merged = pd.merge(df1, df2, on='ID', how='right')
print("\nRight Join Result:")
print(right_merged)
```

```
DataFrame 1:
   ID     Name
0   1    Alice
1   2      Bob
2   3  Charlie

DataFrame 2:
   ID  Age
0   2   25
1   3   30
2   4   22

Inner Join Result:
   ID     Name  Age
0   2      Bob   25
1   3  Charlie   30

Outer Join Result:
   ID     Name   Age
0   1    Alice   NaN
1   2      Bob  25.0
2   3  Charlie  30.0
3   4      NaN  22.0

Left Join Result:
   ID     Name   Age
0   1    Alice   NaN
1   2      Bob  25.0
2   3  Charlie  30.0

Right Join Result:
   ID     Name  Age
0   2      Bob   25
1   3  Charlie   30
2   4      NaN   22
```

## Task 4: Visualization

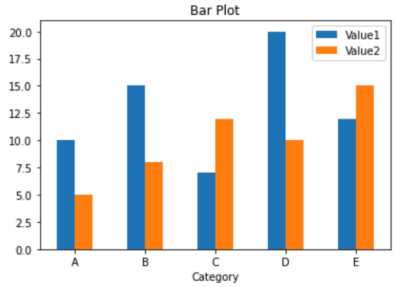
I. Create a bar plot, line plot, and scatter plot using Pandas plotting functions. Customize the plots to make them more informative.

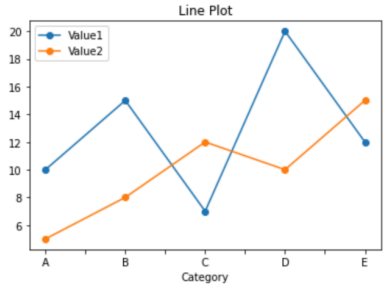
II. Visualize the correlation matrix of numerical columns. Highlight highly correlated features.

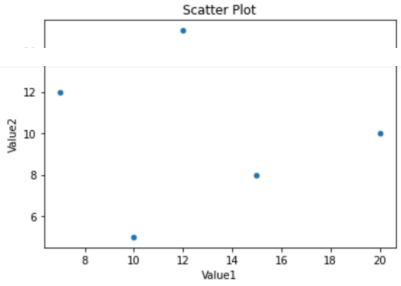
III. Create histograms and box plots for numerical columns. Analyze the distribution and presence of outliers

[53]:
```
data = {'Category': ['A', 'B', 'C', 'D', 'E'],
        'Value1': [10, 15, 7, 20, 12],
        'Value2': [5, 8, 12, 10, 15]}

df = pd.DataFrame(data)

# Bar plot
df.plot(kind='bar', x='Category', y=['Value1', 'Value2'], rot=0, title='Bar Plot')
plt.show()

# Line plot
```

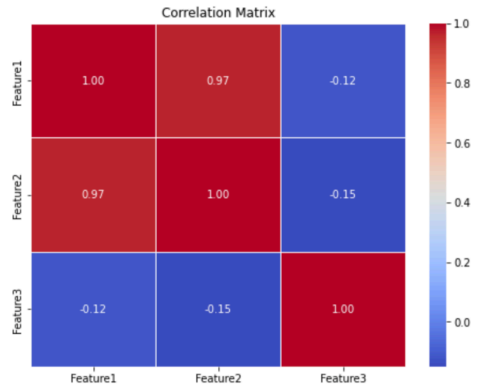
```
df.plot(kind='line', x='Category', y=['Value1', 'Value2'], marker='o', title='Line Plot')
plt.show()

# Scatter plot
df.plot(kind='scatter', x='Value1', y='Value2', title='Scatter Plot')
plt.show()
```



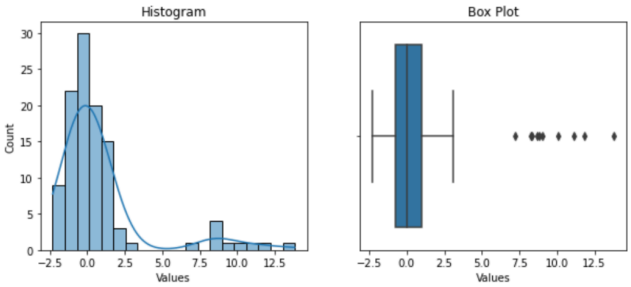




[54]:
```
# II. Visualize the correlation matrix

# Generate some correlated data for illustration
np.random.seed(42)
correlated_data = np.random.randn(100, 3)
correlated_data[:, 1] = 2 * correlated_data[:, 0] + np.random.randn(100) * 0.5

correlation_matrix = pd.DataFrame(correlated_data, columns=['Feature1', 'Feature2', 'Feature3']).corr()

# Heatmap of the correlation matrix
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", linewidths=.5)
plt.title('Correlation Matrix')
plt.show()
```



[55]:
```
# III. Create histograms and box plots

# Sample data with outliers
outliers_data = pd.DataFrame({'Values': np.concatenate([np.random.normal(0, 1, 100), np.random.normal(10, 2, 10)])})

# Histogram
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
sns.histplot(outliers_data['Values'], bins=20, kde=True)
plt.title('Histogram')

# Box plot
plt.subplot(1, 2, 2)
sns.boxplot(x=outliers_data['Values'])
plt.title('Box Plot')

plt.show()
```




## Exploring Numpy

### Task 5: Basic NumPy Operations

1. Create a NumPy array 'arr' with values from 1 to 10.
2. Create another NumPy array 'arr2' with values from 11 to 20.
3. Add, subtract, multiply, and divide 'arr' and 'arr2'. Print the results.

[58]:
```
# 1. Create a NumPy array 'arr' with values from 1 to 10.
arr = np.arange(1, 11)
print("Array 'arr':", arr)
```

```
Array 'arr': [ 1  2  3  4  5  6  7  8  9 10]
```

[59]:
```
# 2. Create another NumPy array 'arr2' with values from 11 to 20.
arr2 = np.arange(11, 21)
print("Array 'arr2':", arr2)
```

```
Array 'arr2': [11 12 13 14 15 16 17 18 19 20]
```

[60]:
```
# 3. Perform arithmetic operations and print the results.
addition_result = arr + arr2
subtraction_result = arr - arr2
multiplication_result = arr * arr2
division_result = arr / arr2

print("Addition Result:", addition_result)
print("Subtraction Result:", subtraction_result)
print("Multiplication Result:", multiplication_result)
print("Division Result:", division_result)
```

```
Addition Result: [12 14 16 18 20 22 24 26 28 30]
Subtraction Result: [-10 -10 -10 -10 -10 -10 -10 -10 -10 -10]
Multiplication Result: [ 11  24  39  56  75  96 119 144 171 200]
Division Result: [0.09090909 0.16666667 0.23076923 0.28571429 0.33333333 0.375
 0.41176471 0.44444444 0.47368421 0.5       ]
```

### Task 6: Array Manipulation ¶

1. Reshape 'arr' into a 2x5 matrix.
2. Transpose the matrix obtained in the previous step.
3. Flatten the transposed matrix into a 1D array.
4. Stack 'arr' and 'arr2' vertically. Print the result.

[61]:
```
# 1. Reshape 'arr' into a 2x5 matrix.
arr_reshaped = arr.reshape(2, 5)
print("1. Reshaped Array (2x5):")
print(arr_reshaped)
```

```
1. Reshaped Array (2x5):
[[ 1  2  3  4  5]
 [ 6  7  8  9 10]]
```

[62]:
```
# 2. Transpose the matrix obtained in the previous step.
arr_transposed = arr_reshaped.T
print("\n2. Transposed Array:")
print(arr_transposed)
```

```
2. Transposed Array:
[[ 1  6]
 [ 2  7]
 [ 3  8]
 [ 4  9]
 [ 5 10]]
```

[63]:
```
# 3. Flatten the transposed matrix into a 1D array.
arr_flattened = arr_transposed.flatten()
print("\n3. Flattened Transposed Array (1D):", arr_flattened)
```

```
3. Flattened Transposed Array (1D): [ 1  6  2  7  3  8  4  9  5 10]
```

[64]:
```
# 4. Stack 'arr' and 'arr2' vertically. Print the result.
stacked_vertical = np.vstack((arr, arr2))
print("\n4. Vertically Stacked Arrays:")
print(stacked_vertical)
```

```
4. Vertically Stacked Arrays:
[[ 1  2  3  4  5  6  7  8  9 10]
 [11 12 13 14 15 16 17 18 19 20]]
```

## Task 7: Statistical Operations

1. Calculate the mean, median, and standard deviation of 'arr'.
2. Find the maximum and minimum values in 'arr'.
3. Normalize 'arr' (subtract the mean and divide by the standard deviation).

[67]:
```
# Calculate mean, median, and standard deviation of 'arr'
arr_mean = np.mean(arr)
arr_median = np.median(arr)
arr_std = np.std(arr)
print("Mean of 'arr':", arr_mean)
print("Median of 'arr':", arr_median)
print("Standard Deviation of 'arr':", arr_std)
```

```
Mean of 'arr': 5.5
Median of 'arr': 5.5
Standard Deviation of 'arr': 2.8722813232690143
```

[68]:
```
# Find maximum and minimum values in 'arr'
arr_max = np.max(arr)
arr_min = np.min(arr)
print("Maximum Value in 'arr':", arr_max)
print("Minimum Value in 'arr':", arr_min)
```

```
Maximum Value in 'arr': 10
Minimum Value in 'arr': 1
```

[69]:
```
# Normalize 'arr' (subtract the mean and divide by the standard deviation)
arr_normalized = (arr - arr_mean) / arr_std
print("\nNormalized 'arr':", arr_normalized)
```

```
Normalized 'arr': [-1.5666989  -1.21854359 -0.87038828 -0.52223297 -0.17407766  0.17407766
  0.52223297  0.87038828  1.21854359  1.5666989 ]
```

## Task 8: Boolean Indexing

1. Create a boolean array 'bool_arr' for elements in 'arr' greater than 5.
2. Use 'bool_arr' to extract the elements from 'arr' that are greater than 5.

[70]:
```
import numpy as np

# Create a NumPy array 'arr' with values from 1 to 10.
arr = np.arange(1, 11)
print("Original 'arr':", arr)

# Create a boolean array 'bool_arr' for elements in 'arr' greater than 5.
bool_arr = arr > 5
print("\nBoolean array 'bool_arr':", bool_arr)
```

```
Original 'arr': [ 1  2  3  4  5  6  7  8  9 10]

Boolean array 'bool_arr': [False False False False False  True  True  True  True  True]
```

[71]:
```
# Use 'bool_arr' to extract elements from 'arr' that are greater than 5.
elements_greater_than_5 = arr[bool_arr]
print("\nElements in 'arr' greater than 5:", elements_greater_than_5)
```

```
Elements in 'arr' greater than 5: [ 6  7  8  9 10]
```

## Task 9: Random Module

1. Generate a 3x3 matrix with random values between 0 and 1.
2. Create an array of 10 random integers between 1 and 100.
3. Shuffle the elements of 'arr' randomly.

[72]:
```
random_matrix = np.random.rand(3, 3)
print("Random 3x3 Matrix:")
print(random_matrix)
```

```
Random 3x3 Matrix:
[[0.41039683 0.98237862 0.1120389 ]
 [0.3978556  0.96947043 0.86550713]
 [0.81707207 0.25790283 0.17088759]]
```

[73]:
```
random_integers = np.random.randint(1, 101, 10)
print("\nArray of 10 Random Integers:")
print(random_integers)
```

```
Array of 10 Random Integers:
[51 62 57 66 79 75  8 26 51 45]
```

[74]:
```
arr = np.array([1, 2, 3, 4, 5])
np.random.shuffle(arr)
print("\nShuffled 'arr':", arr)
```

```
Shuffled 'arr': [5 3 2 1 4]
```

## Task 10: Universal Functions (ufunc)

1. Apply the square root function to all elements in 'arr'.
2. Use the exponential function to calculate exex for each element in 'arr'.

[75]:
```
arr_sqrt = np.sqrt(arr)
print("\nSquare Root of 'arr':", arr_sqrt)
```

```
Square Root of 'arr': [2.23606798 1.73205081 1.41421356 1.         2.        ]
```

[76]:
```
arr_exp = np.exp(arr)
print("Exponential of 'arr':", arr_exp)
```

```
Exponential of 'arr': [148.4131591   20.08553692   7.3890561    2.71828183  54.59815003]
```

## Task 11: Linear Algebra Operations

1. Create a 3x3 matrix 'mat_a' with random values.
2. Create a 3x1 matrix 'vec_b' with random values.
3. Multiply 'mat_a' and 'vec_b' using the dot product.

[77]:
```
mat_a = np.random.rand(3, 3)
print("\nRandom 3x3 Matrix 'mat_a':")
print(mat_a)
```

```
Random 3x3 Matrix 'mat_a':
[[0.32367924 0.42543644 0.50761038]
 [0.24240973 0.11483682 0.61062004]
 [0.28863055 0.58123822 0.15436272]]
```

[78]:
```
vec_b = np.random.rand(3, 1)
print("\nRandom 3x1 Matrix 'vec_b':")
print(vec_b)
```

```
Random 3x1 Matrix 'vec_b':
[[0.4811401 ]
 [0.53258943]
 [0.05182354]]
```

[79]:
```
result_dot_product = np.dot(mat_a, vec_b)
print("\nResult of Matrix Multiplication (Dot Product):")
print(result_dot_product)
```

```
Result of Matrix Multiplication (Dot Product):
[[0.40862418]
 [0.20943841]
 [0.45643269]]
```

## Task 12: Broadcasting

1. Create a 2D array 'matrix' with values from 1 to 9.
2. Subtract the mean of each row from each element in that row.

[80]:
```
matrix = np.arange(1, 10).reshape(3, 3)
print("\n2D Array 'matrix':")
print(matrix)
```

```
2D Array 'matrix':
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

[81]:
```
mean_row = matrix.mean(axis=1, keepdims=True)
result_broadcasting = matrix - mean_row
print("\nResult after Broadcasting (subtracting mean of each row):")
print(result_broadcasting)
```

```
Result after Broadcasting (subtracting mean of each row):
[[-1.  0.  1.]
 [-1.  0.  1.]
 [-1.  0.  1.]]
```

[ ]:

# Experiment 2
# Wine Quality Prediction

---

[4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb



from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.linear_model import LogisticRegression
```

[8]:
```
wine_data =  pd.read_csv("WineQT.csv")
```

[9]:
```
wine_data.head()
```

[9]:

| | fixed acidity | volatile acidity | citric acid | residual sugar | chlorides | free sulfur dioxide | total sulfur dioxide | density | pH | sulphates | alcohol | quality | Id |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 | 0 |
| **1** | 7.8 | 0.88 | 0.00 | 2.6 | 0.098 | 25.0 | 67.0 | 0.9968 | 3.20 | 0.68 | 9.8 | 5 | 1 |
| **2** | 7.8 | 0.76 | 0.04 | 2.3 | 0.092 | 15.0 | 54.0 | 0.9970 | 3.26 | 0.65 | 9.8 | 5 | 2 |
| **3** | 11.2 | 0.28 | 0.56 | 1.9 | 0.075 | 17.0 | 60.0 | 0.9980 | 3.16 | 0.58 | 9.8 | 6 | 3 |
| **4** | 7.4 | 0.70 | 0.00 | 1.9 | 0.076 | 11.0 | 34.0 | 0.9978 | 3.51 | 0.56 | 9.4 | 5 | 4 |

[11]:
```
wine_data.describe().T
```

[11]:

| | count | mean | std | min | 25% | 50% | 75% | max |
|---|---|---|---|---|---|---|---|---|
| **fixed acidity** | 1143.0 | 8.311111 | 1.747595 | 4.60000 | 7.10000 | 7.90000 | 9.100000 | 15.90000 |
| **volatile acidity** | 1143.0 | 0.531339 | 0.179633 | 0.12000 | 0.39250 | 0.52000 | 0.640000 | 1.58000 |
| **citric acid** | 1143.0 | 0.268364 | 0.196686 | 0.00000 | 0.09000 | 0.25000 | 0.420000 | 1.00000 |
| **residual sugar** | 1143.0 | 2.532152 | 1.355917 | 0.90000 | 1.90000 | 2.20000 | 2.600000 | 15.50000 |
| **chlorides** | 1143.0 | 0.086933 | 0.047267 | 0.01200 | 0.07000 | 0.07900 | 0.090000 | 0.61100 |
| **free sulfur dioxide** | 1143.0 | 15.615486 | 10.250486 | 1.00000 | 7.00000 | 13.00000 | 21.000000 | 68.00000 |
| **total sulfur dioxide** | 1143.0 | 45.914698 | 32.782130 | 6.00000 | 21.00000 | 37.00000 | 61.000000 | 289.00000 |
| **density** | 1143.0 | 0.996730 | 0.001925 | 0.99007 | 0.99557 | 0.99668 | 0.997845 | 1.00369 |
| **pH** | 1143.0 | 3.311015 | 0.156664 | 2.74000 | 3.20500 | 3.31000 | 3.400000 | 4.01000 |
| **sulphates** | 1143.0 | 0.657708 | 0.170399 | 0.33000 | 0.55000 | 0.62000 | 0.730000 | 2.00000 |
| **alcohol** | 1143.0 | 10.442111 | 1.082196 | 8.40000 | 9.50000 | 10.20000 | 11.100000 | 14.90000 |
| **quality** | 1143.0 | 5.657043 | 0.805824 | 3.00000 | 5.00000 | 6.00000 | 6.000000 | 8.00000 |
| **Id** | 1143.0 | 804.969379 | 463.997116 | 0.00000 | 411.00000 | 794.00000 | 1209.500000 | 1597.00000 |

[13]:
```
wine_data.isnull().sum() #pre-cleaned dataset
```

[13]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
Id                      0
dtype: int64
```

[19]:
```
wine_data['best quality'] = [1 if x > 5 else 0 for x in wine_data.quality]
wine_data.replace({'white': 1, 'red': 0}, inplace=True)
```

[20]:
```
features = wine_data.drop(['quality', 'best quality'], axis=1)
target = wine_data['best quality']

xtrain, xtest, ytrain, ytest = train_test_split(
    features, target, test_size=0.2, random_state=40)
```

```
xtrain.shape, xtest.shape
```

[20]:
```
((914, 12), (229, 12))
```

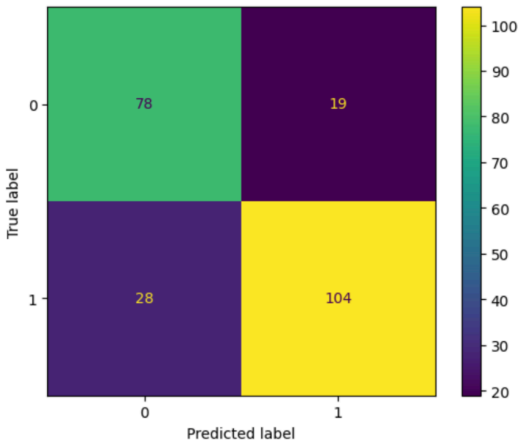
[21]:
```
norm = MinMaxScaler()
xtrain = norm.fit_transform(xtrain)
xtest = norm.transform(xtest)
```

[22]:
```
models = [LogisticRegression(), XGBClassifier(), SVC(kernel='rbf')]

for i in range(3):
    models[i].fit(xtrain, ytrain)

    print(f'{models[i]} : ')
    print('Training Accuracy : ', metrics.roc_auc_score(ytrain, models[i].predict(xtrain)))
    print('Validation Accuracy : ', metrics.roc_auc_score(
        ytest, models[i].predict(xtest)))
    print()
```

```
LogisticRegression() :
Training Accuracy :  0.7573006134969325
Validation Accuracy :  0.7523430178069354

XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, feature_types=None, gamma=0, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_bin=256, max_cat_threshold=64, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=0, ...) :
Training Accuracy :  1.0
Validation Accuracy :  0.796001249609497

SVC() :
Training Accuracy :  0.7709009984361843
Validation Accuracy :  0.7520306154326772
```

[23]:
```
metrics.plot_confusion_matrix(models[1], xtest, ytest)
plt.show()
```

```
c:\python\lib\site-packages\sklearn\utils\deprecation.py:87: FutureWarning: Function plot_confusion_matrix is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimator.
  warnings.warn(msg, category=FutureWarning)
```




[25]:
```
print(metrics.classification_report(ytest,models[1].predict(xtest)))
```

```
              precision    recall  f1-score   support

           0       0.74      0.80      0.77        97
           1       0.85      0.79      0.82       132

    accuracy                           0.79       229
   macro avg       0.79      0.80      0.79       229
weighted avg       0.80      0.79      0.80       229
```

# Experiment 3
# House Price prediction

---

[ ]:
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import OneHotEncoder
```

Reading dataset

[ ]:
```
data = pd.read_csv("Housing.csv")
data.head(5)
```

[6]:

| | price | area | bedrooms | bathrooms | stories | mainroad | guestroom | basement | hotwaterheating | airconditioning | parking | prefarea | furnishingstatus |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 13300000 | 7420 | 4 | 2 | 3 | yes | no | no | no | yes | 2 | yes | furnished |
| **1** | 12250000 | 8960 | 4 | 4 | 4 | yes | no | no | no | yes | 3 | no | furnished |
| **2** | 12250000 | 9960 | 3 | 2 | 2 | yes | no | yes | no | no | 2 | yes | semi-furnished |
| **3** | 12215000 | 7500 | 4 | 2 | 2 | yes | no | yes | no | yes | 3 | yes | furnished |
| **4** | 11410000 | 7420 | 4 | 1 | 2 | yes | yes | yes | no | yes | 2 | no | furnished |

[ ]:
```
# Separate features and target variable
X = data.drop('price', axis=1)
y = data['price']
```

[ ]:
```
# One-Hot Encode categorical variables
X_encoded = pd.get_dummies(X, columns=['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'airconditioning', 'prefarea', 'furnishingstatu

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, test_size=0.2, random_state=42)

# Create a linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)
```

```
# Make predictions on the test set
y_pred = model.predict(X_test)
```

[ ]:
```
# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error: {mse}')
```

```
Mean Squared Error: 1754318687330.6638
```

[ ]:
```
# Visualize predictions vs. actual values
plt.scatter(y_test, y_pred)
plt.xlabel("Actual Prices")
plt.ylabel("Predicted Prices")
plt.title("Actual Prices vs. Predicted Prices")
plt.show()
```

[ ]:
```python
# Example usage: predict the price for a new house
new_house_features = pd.DataFrame({
    'area': [7420],
    'bedrooms': [4],
    'bathrooms': [2],
    'stories': [3],
    'mainroad': ['yes'],  # Assuming 'yes' for the rest of the categorical features
    'guestroom': ['yes'],
    'basement': ['no'],
    'hotwaterheating': ['no'],
    'airconditioning': ['yes'],
    'parking': [2],
    'prefarea': ['yes'],
    'furnishingstatus': ['furnished']
})

# One-Hot Encode categorical variables for new house features
new_house_features_encoded = pd.get_dummies(new_house_features, columns=['mainroad', 'guestroom', 'basement', 'hotwaterheating', 'aircondition

# Ensure all necessary columns are present for prediction
missing_columns = set(X_train.columns) - set(new_house_features_encoded.columns)
for col in missing_columns:
    new_house_features_encoded[col] = 0  # Add missing columns with value 0

# Reorder columns to match the order during training
new_house_features_encoded = new_house_features_encoded[X_train.columns]
```

[ ]:
```python
# Make predictions for the new house
predicted_price = model.predict(new_house_features_encoded)
print(f'Predicted Price for the new house: {predicted_price[0]}')
```

```
Predicted Price for the new house: 6179038.87755126
```

# Experiment 4
# Air Quality Prediction

---

[ ]:
```
import pandas as pd
import numpy as np
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.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

data = pd.read_csv('/content/city_day.csv')


data.dropna(inplace=True)


print(data.describe())
```

```
              PM2.5         PM10           NO          NO2          NOx  \
count  6236.000000  6236.000000  6236.000000  6236.000000  6236.000000
mean     61.327365   123.418321    17.015191    31.708190    32.448956
std      53.709682    85.791491    20.037836    18.784041    27.388129
min       2.000000     7.800000     0.250000     0.170000     0.170000
25%      27.927500    66.970000     5.080000    15.977500    14.547500
50%      47.490000   103.010000    10.060000    28.900000    24.285000
75%      73.442500   150.770000    19.392500    43.632500    39.622500
max     639.190000   796.880000   159.220000   140.170000   224.090000

               NH3           CO          SO2           O3      Benzene  \
count  6236.000000  6236.000000  6236.000000  6236.000000  6236.000000
mean     20.737070     0.984344    11.514426    36.127691     3.700361
std      16.088215     1.356161     7.166113    19.553695     5.062159
min       0.120000     0.000000     0.710000     1.550000     0.000000
25%      10.390000     0.490000     6.557500    22.357500     0.910000
50%      14.690000     0.730000     9.875000    32.540000     2.435000
75%      28.545000     1.060000    14.430000    45.512500     4.620000
max     166.700000    16.230000    70.390000   162.330000    64.440000


           Toluene       Xylene          AQI
count  6236.000000  6236.000000  6236.000000
mean     10.323696     2.557439   140.510103
std      12.287223     4.535060    92.738826
min       0.000000     0.000000    23.000000
25%       2.210000     0.300000    78.000000
50%       6.310000     1.250000   112.000000
75%      13.040000     3.030000   166.000000
max     103.000000   125.180000   677.000000
```

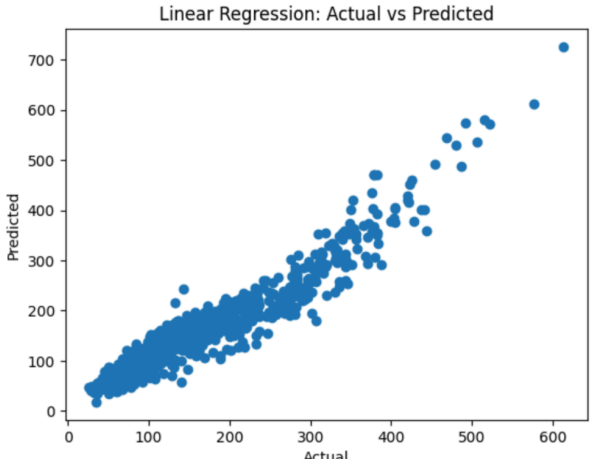
[ ]:
```

relevant_features = ['PM2.5', 'PM10', 'NO2', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene', 'City', 'Date']

data['Pollutants_Average'] = data[['PM2.5', 'PM10', 'NO2', 'CO', 'SO2', 'O3']].mean(axis=1)
```

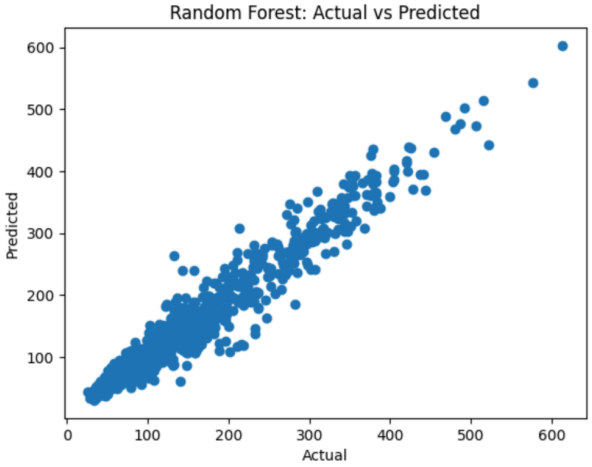
[ ]:
```
X = data[['PM2.5', 'PM10', 'NO2', 'CO', 'SO2', 'O3', 'Benzene', 'Toluene', 'Xylene']]
y = data['AQI']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

model_lr = LinearRegression()
model_rf = RandomForestRegressor()

model_lr.fit(X_train, y_train)
model_rf.fit(X_train, y_train)

y_pred_lr = model_lr.predict(X_test)
print("Linear Regression Metrics:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_lr))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_lr))
print("R-squared:", r2_score(y_test, y_pred_lr))

# Random Forest
y_pred_rf = model_rf.predict(X_test)
print("\nRandom Forest Metrics:")
print("Mean Squared Error:", mean_squared_error(y_test, y_pred_rf))
print("Mean Absolute Error:", mean_absolute_error(y_test, y_pred_rf))
print("R-squared:", r2_score(y_test, y_pred_rf))
```

```
Linear Regression Metrics:
Mean Squared Error: 679.1269034340863
```

```
Mean Absolute Error: 18.575630319710974
R-squared: 0.9183762350621478

Random Forest Metrics:
Mean Squared Error: 451.761092147436
Mean Absolute Error: 14.47036858974359
R-squared: 0.9457031653332393
```

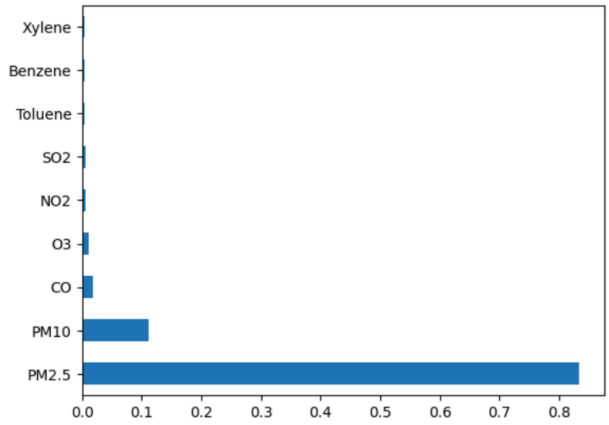
[ ]:
```
plt.scatter(y_test, y_pred_lr)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Linear Regression: Actual vs Predicted")
plt.show()

plt.scatter(y_test, y_pred_rf)
plt.xlabel("Actual")
plt.ylabel("Predicted")
plt.title("Random Forest: Actual vs Predicted")
plt.show()
```



[ ]:
```
print("Linear Regression Coefficients:")
print(model_lr.coef_)


print("\nRandom Forest Feature Importances:")
print(model_rf.feature_importances_)


feat_importances = pd.Series(model_rf.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
Linear Regression Coefficients:
[ 0.80315178  0.47070381  0.02502601  8.38779514 -0.03778825  0.26764177
  0.27866611 -0.12764135 -0.24244239]

Random Forest Feature Importances:
[0.83351888 0.11184619 0.00619749 0.01781194 0.00499727 0.01174644
 0.00468331 0.00476075 0.00443774]
```



# Experiment 5
# Credit Card fault prediction

---

[1]:
```
# imports
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

pd.set_option('display.float_format', lambda x: '%.2f' % x)

sns.set_theme()

df = pd.read_csv('/kaggle/input/creditcardfraud/creditcard.csv')

# turn off warnings
import warnings
warnings.filterwarnings('ignore')
```

## Data Set

- 0 => legit transaction
- 1 => fraud

[2]:
```
df.sample(5)
```

[2]:

| | Time | V1 | V2 | V3 | V4 | V5 | V6 | V7 | V8 | V9 | ... | V21 | V22 | V23 | V24 | V25 | V26 | V27 | V28 | Amount | Class |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **49166** | 43942.00 | 1.22 | 0.66 | -0.45 | 0.80 | 0.13 | -1.23 | 0.36 | -0.22 | -0.54 | ... | -0.08 | -0.18 | -0.08 | 0.44 | 0.54 | 0.34 | -0.03 | 0.04 | 0.76 | 0 |
| **283877** | 171944.00 | 2.06 | 0.02 | -1.04 | 0.41 | -0.06 | -1.20 | 0.25 | -0.39 | 0.40 | ... | -0.28 | -0.62 | 0.33 | 0.07 | -0.27 | 0.19 | -0.06 | -0.06 | 0.89 | 0 |
| **222314** | 142926.00 | -0.74 | 1.19 | 0.84 | -0.39 | 0.87 | -0.12 | 0.93 | -0.02 | -0.82 | ... | 0.28 | 0.92 | -0.60 | 0.77 | 0.95 | 0.15 | 0.15 | 0.12 | 2.70 | 0 |
| **272641** | 165194.00 | 1.86 | -1.13 | -1.50 | -0.33 | -0.53 | -0.60 | -0.18 | -0.24 | -0.53 | ... | 0.07 | 0.50 | -0.17 | -0.39 | 0.18 | -0.03 | -0.02 | -0.04 | 158.00 | 0 |
| **90270** | 62947.00 | -2.05 | -0.18 | -0.15 | 1.19 | 0.76 | 0.02 | -0.63 | 0.95 | -0.63 | ... | 0.16 | -0.00 | -0.60 | -1.29 | 0.14 | -0.17 | -0.13 | -0.46 | 60.00 | 0 |

5 rows × 31 columns

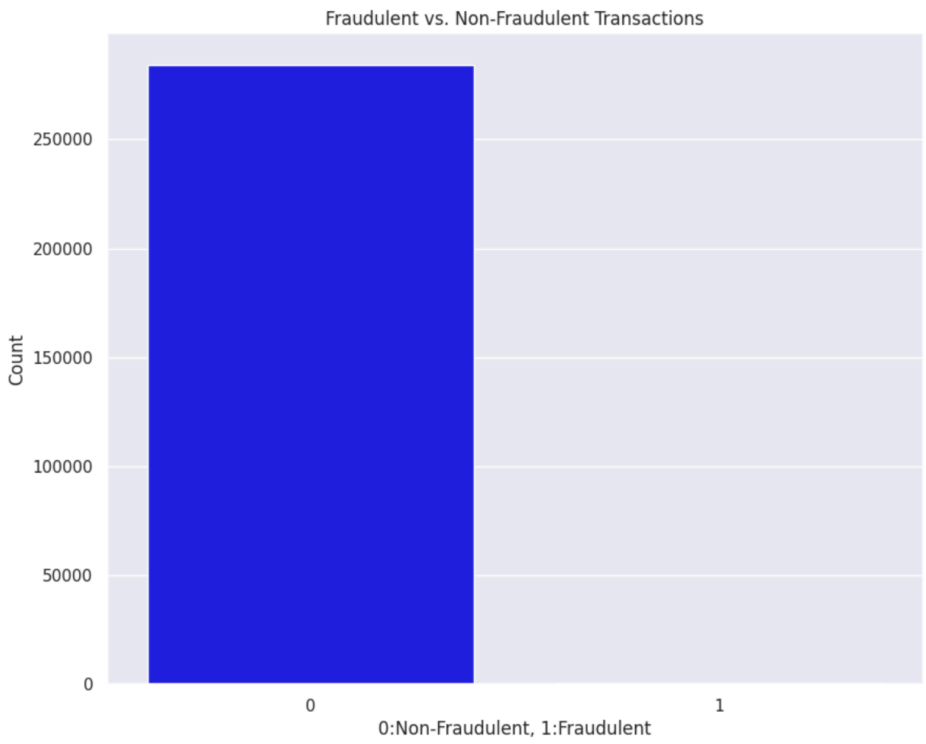
[3]:
```
df.loc[:, ['Time', 'Amount']].describe()
```

[3]:

| | Time | Amount |
|---|---|---|
| **count** | 284807.00 | 284807.00 |
| **mean** | 94813.86 | 88.35 |
| **std** | 47488.15 | 250.12 |
| **min** | 0.00 | 0.00 |
| **25%** | 54201.50 | 5.60 |
| **50%** | 84692.00 | 22.00 |
| **75%** | 139320.50 | 77.16 |
| **max** | 172792.00 | 25691.16 |

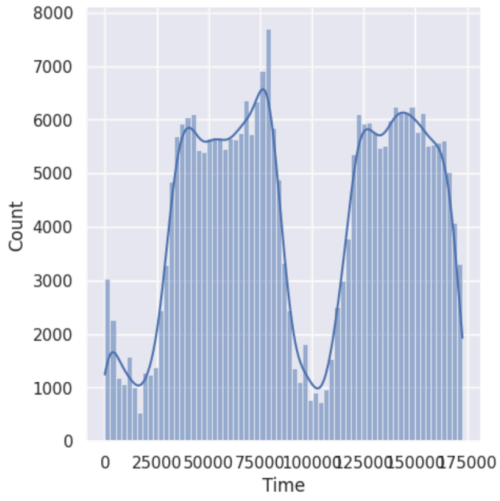
[4]:
```
plt.figure(figsize=(10,8))
sns.barplot(x=df['Class'].value_counts().index, y=df['Class'].value_counts(), color='blue')
plt.title('Fraudulent vs. Non-Fraudulent Transactions')
plt.ylabel('Count')
plt.xlabel('0:Non-Fraudulent, 1:Fraudulent')
```

[4]:
```
Text(0.5, 0, '0:Non-Fraudulent, 1:Fraudulent')
```




[5]:
```
sns.displot(df['Time'], kde=True)
```

[5]:
```
<seaborn.axisgrid.FacetGrid at 0x7f7f583f53f0>
```



# Experiment 6
# Hypothesis Testing (t test)

---

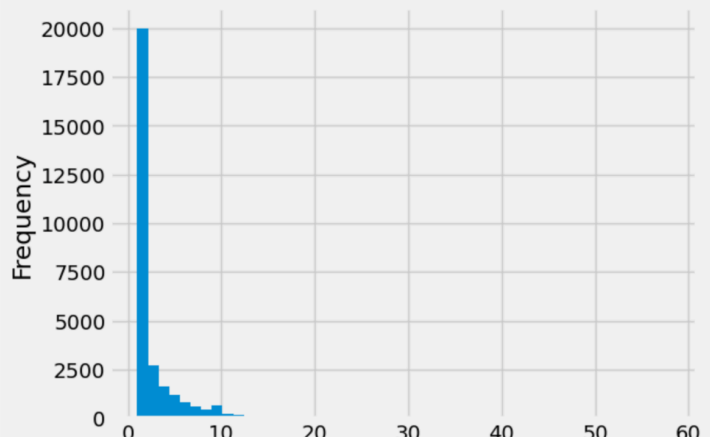
## Hypothesis testing with firearm licenses

[10]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as stats
```

[7]:
```
plt.style.use('fivethirtyeight')
licensees = pd.read_csv("/content/federal-firearm-licensees.csv")
licensees['Premise Zip Code'].value_counts().plot.hist(bins=50)
```

```
<ipython-input-7-b9e5de403f6f>:2: DtypeWarning: Columns (1,2,3,4,6,12,17) have mixed types. Specify dtype option on import or set low_memory=
False.
  licensees = pd.read_csv("/content/federal-firearm-licensees.csv")
```

[7]:
```
<Axes: ylabel='Frequency'>
```




[8]:
```
licensees['Premise Zip Code'].value_counts().mean()
```

[8]:
```
2.756617394293572
```

[9]:
```
X = licensees['Premise Zip Code'].value_counts()
```

[11]:
```
def t_value(X, h_0):
    se = np.sqrt(np.var(X) / len(X))
    return (np.mean(X) - h_0) / se

def p_value(t):
    # Two-sided p-value, so we multiply by 2.
    return stats.norm.sf(abs(t))*2

t = t_value(X, 2.75)
p = p_value(t)
```

[12]:
```
t, p
```

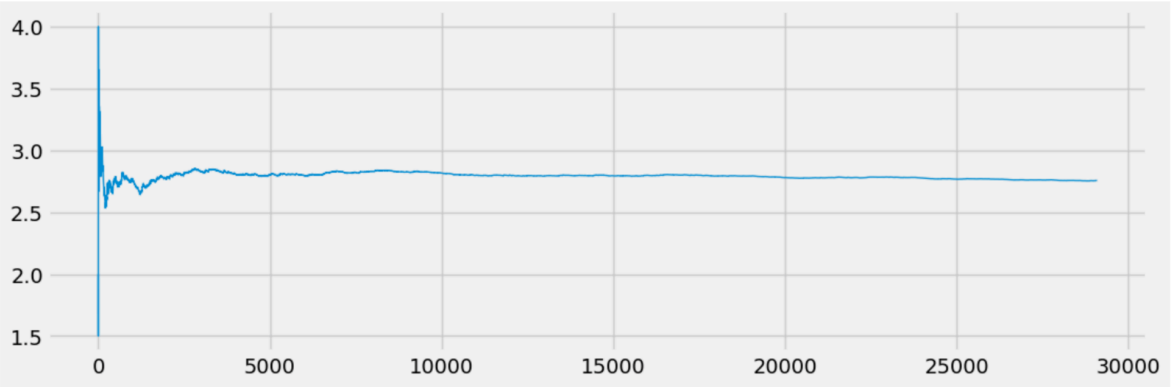
[12]:
```
(0.35702376432918465, 0.7210740152174941)
```

[13]:
```
import scipy.stats as stats

stats.ttest_1samp(a=X, popmean=2.75)
```

[13]:
```
TtestResult(statistic=0.35701762773873, pvalue=0.7210811989731936, df=29089)
```

[14]:
```
r = (licensees['Premise Zip Code']
        .value_counts()
        .sample(len(licensees['Premise Zip Code'].unique()) - 1))
pd.Series(r.cumsum() / np.array(range(1, len(r) + 1))).reset_index(drop=True).plot.line(
    figsize=(12, 4), linewidth=1
)
```

[14]: <Axes: >



# Experiment 7
# Hypothesis Testing (ANOVA)

---

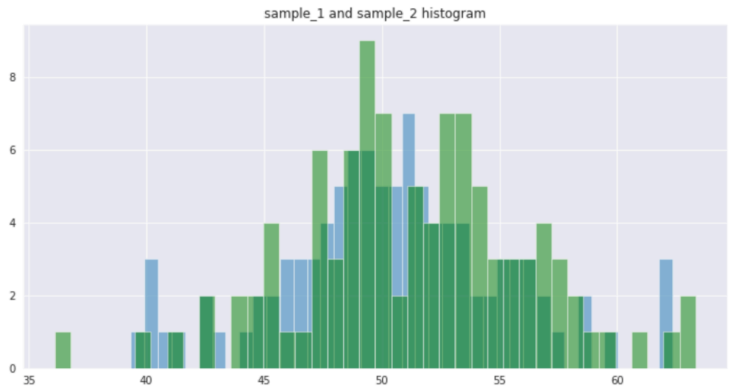
[1]:
```
import numpy as np
import numpy.random as random
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
sns.set_style("darkgrid")

plt.rcParams['figure.figsize'] = (12, 6)
```

## Create data for testing

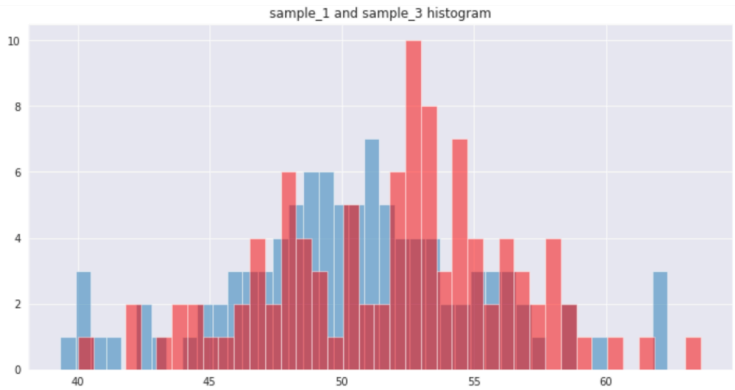
[2]:
```
random.seed(10)

sample_1 = 5 * random.randn(100) + 50
sample_2 = 5 * random.randn(100) + 51
sample_3 = 5 * random.randn(100) + 52

print("sample_1: mean = %.3f, std = %.3f" % (np.mean(sample_1), np.std(sample_1)))
print("sample_2: mean = %.3f, std = %.3f" % (np.mean(sample_2), np.std(sample_2)))
print("sample_3: mean = %.3f, std = %.3f" % (np.mean(sample_3), np.std(sample_3)))
```

```
sample_1: mean = 50.397, std = 4.835
sample_2: mean = 51.346, std = 4.927
sample_3: mean = 51.880, std = 4.567
```

[3]:
```
plt.hist(sample_1, bins = 40, alpha = 0.5)
plt.hist(sample_2, bins = 40, alpha = 0.5, color = "green")
plt.title("sample_1 and sample_2 histogram")
plt.show()
```




[4]:
```
plt.hist(sample_1, bins = 40, alpha = 0.5)
plt.hist(sample_3, bins = 40, alpha = 0.5, color = "red")
plt.title("sample_1 and sample_3 histogram")
plt.show()
```



## Student's t-Test

📌 - Let's first compare sample_1 and sample_2:

- sample_1: mean = 50.397, std = 4.835
- sample_2: mean = 51.346, std = 4.927

[5]:
```
stat, p_value = stats.ttest_ind(sample_1, sample_2)
print("Statistics = %.3f, p-value = %.3f" % (stat, p_value))

alpha = 0.05

if p_value > alpha:
    print("Same distributions (fail to reject H0)")
else:
    print("Different distributions (reject H0)")
```

```
Statistics = -1.368, p-value = 0.173
Same distributions (fail to reject H0)
```

The p-value is not small enough to reject the null hypothesis in this instance even though we created the sample means to be slightly different.

📌 - Now we can compare sample_1 and sample_3:

- sample_1: mean = 50.397, std = 4.835
- sample_3: mean = 51.880, std = 4.567

[6]:
```
stat, p_value = stats.ttest_ind(sample_1, sample_3)
print("Statistics = %.3f, p-value = %.3f" % (stat, p_value))

alpha = 0.05

if p_value > alpha:
    print("Same distributions (fail to reject H0)")
else:
    print("Different distributions (reject H0)")
```

```
Statistics = -2.219, p-value = 0.028
Different distributions (reject H0)
```

The p-value is now small enough to reject the null hypothesis in this instance.

## Paired Student's t-Test

A dependent samples t-test is also used to compare two means on a single dependent variable. Unlike the independent samples test, however, a dependent samples t-test is used to compare the means of a single sample or of two matched or paired samples. The paired Student's t-test can be implemented using the ttest_rel() SciPy function.

## Analysis of Variance Test (ANOVA)

ANOVA is a statistical test that assumes that the mean across 2 or more groups are equal. If the evidence suggests that this is not the case, the null hypothesis is rejected and at least one data sample has a different distribution.

- Fail to Reject H0: All sample distributions are equal.
- Reject H0: One or more sample distributions are not equal.

Importantly, the test can only comment on whether all samples are the same or not; it cannot quantify which samples differ or by how much.

[7]:
```
random.seed(20)

sample_4 = 5 * random.randn(100) + 50
sample_5 = 5 * random.randn(100) + 50
sample_6 = 5 * random.randn(100) + 52

print("sample_4: mean = %.3f, std = %.3f" % (np.mean(sample_4), np.std(sample_4)))
print("sample_5: mean = %.3f, std = %.3f" % (np.mean(sample_5), np.std(sample_5)))
print("sample_6: mean = %.3f, std = %.3f" % (np.mean(sample_6), np.std(sample_6)))
print("-" * 35)

stat, p_value = stats.f_oneway(sample_4, sample_5, sample_6)
print("Statistics = %.3f, p-value = %.3f" % (stat, p_value))

alpha = 0.05

if p_value > alpha:
    print("Same distributions (fail to reject H0)")
```

```
    print("Same distributions (fail to reject H0)")
else:
    print("Different distributions (reject H0)")
```

```
sample_4: mean = 49.727, std = 5.363
sample_5: mean = 50.433, std = 5.000
sample_6: mean = 51.814, std = 4.423
-----------------------------------
Statistics = 4.563, p-value = 0.011
Different distributions (reject H0)
```

# Experiment 8
# Implement KNN and other models and compare them on a dataset.

---

[4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
```

[5]:
```
df_drug = pd.read_csv("drug200.csv")
```

[6]:
```
df_drug.head()
```

[6]:

| | Age | Sex | BP | Cholesterol | Na_to_K | Drug |
|---|---|---|---|---|---|---|
| **0** | 23 | F | HIGH | HIGH | 25.355 | DrugY |
| **1** | 47 | M | LOW | HIGH | 13.093 | drugC |
| **2** | 47 | M | LOW | HIGH | 10.114 | drugC |
| **3** | 28 | F | NORMAL | HIGH | 7.798 | drugX |
| **4** | 61 | F | LOW | HIGH | 18.043 | DrugY |

[7]:
```
print(df_drug.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 6 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Age          200 non-null    int64  
 1   Sex          200 non-null    object 
 2   BP           200 non-null    object 
 3   Cholesterol  200 non-null    object 
 4   Na_to_K      200 non-null    float64
 5   Drug         200 non-null    object 
dtypes: float64(1), int64(1), object(4)
memory usage: 9.5+ KB
None
```

[8]:
```
df_drug.Drug.value_counts()
```

[8]:
```
Drug
DrugY    91
drugX    54
drugA    23
drugC    16
drugB    16
Name: count, dtype: int64
```

[9]:
```
df_drug.describe()
```

[9]:

| | Age | Na_to_K |
|---|---|---|
| **count** | 200.000000 | 200.000000 |
| **mean** | 44.315000 | 16.084485 |
| **std** | 16.544315 | 7.223956 |
| **min** | 15.000000 | 6.269000 |
| **25%** | 31.000000 | 10.445500 |
| **50%** | 45.000000 | 13.936500 |
| **75%** | 58.000000 | 19.380000 |
| **max** | 74.000000 | 38.247000 |

[21]:
```
bin_age = [0, 19, 29, 39, 49, 59, 69, 80]
category_age = ['<20s', '20s', '30s', '40s', '50s', '60s', '>60s']
df_drug['Age_binned'] = pd.cut(df_drug['Age'], bins=bin_age, labels=category_age)
df_drug = df_drug.drop(['Age'], axis = 1)
```

[22]:
```
bin_NatoK = [0, 9, 19, 29, 50]
category_NatoK = ['<10', '10-20', '20-30', '>30']
df_drug['Na_to_K_binned'] = pd.cut(df_drug['Na_to_K'], bins=bin_NatoK, labels=category_NatoK)
df_drug = df_drug.drop(['Na_to_K'], axis = 1)
```

### Splitting the dataset

[23]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
```

[24]:
```
X = df_drug.drop(["Drug"], axis=1)
y = df_drug["Drug"]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
#split in 70/30
```

[25]:
```
X_train = pd.get_dummies(X_train)
X_test = pd.get_dummies(X_test)
```

[26]:
```
X_train.head()
```

[26]:

| | Sex_F | Sex_M | BP_HIGH | BP_LOW | BP_NORMAL | Cholesterol_HIGH | Cholesterol_NORMAL | Age_binned_<20s | Age_binned_20s | Age_binned_30s | Age_binnec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **131** | False | True | False | True | False | False | True | False | False | False | |
| **96** | True | False | False | True | False | True | False | False | False | False | |
| **181** | True | False | False | False | True | True | False | False | False | False | |
| **19** | True | False | True | False | False | False | True | False | False | True | |
| **153** | True | False | False | True | False | False | True | False | False | False | |

[27]:
```
X_test.head()
```

[27]:

| | Sex_F | Sex_M | BP_HIGH | BP_LOW | BP_NORMAL | Cholesterol_HIGH | Cholesterol_NORMAL | Age_binned_<20s | Age_binned_20s | Age_binned_30s | Age_binnec |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **18** | False | True | False | True | False | True | False | False | True | False | |
| **170** | True | False | False | False | True | True | False | False | True | False | |
| **107** | False | True | False | True | False | True | False | False | False | False | |
| **98** | False | True | True | False | False | False | True | False | True | False | |
| **177** | False | True | False | False | True | True | False | False | True | False | |

## 7. Models 🛠️

### 7.1 Logistic Regression

[28]:
```
from sklearn.linear_model import LogisticRegression
LRclassifier = LogisticRegression(solver='liblinear', max_iter=5000)
LRclassifier.fit(X_train, y_train)

y_pred = LRclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
LRAcc = accuracy_score(y_pred,y_test)
print('Logistic Regression accuracy is: {:.2f}%'.format(LRAcc*100))
```

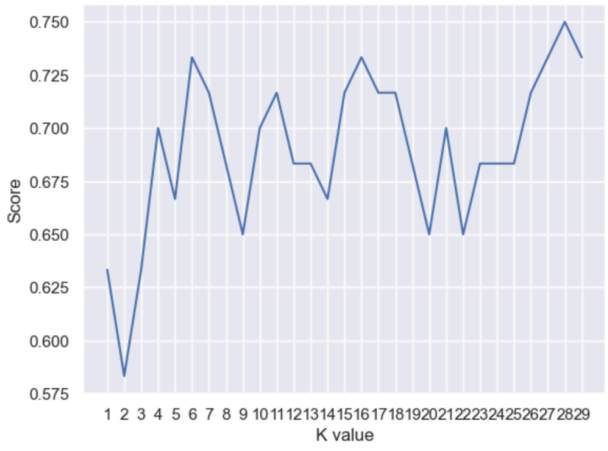
```
              precision    recall  f1-score   support

       DrugY       0.95      0.70      0.81        30
       drugA       0.67      0.80      0.73         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18

    accuracy                           0.83        60
   macro avg       0.77      0.90      0.82        60
weighted avg       0.86      0.83      0.83        60

[[21  2  1  2  4]
 [ 1  4  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0 18]]
Logistic Regression accuracy is: 83.33%
```

## 7.2 K Neighbours

[29]:
```
from sklearn.neighbors import KNeighborsClassifier
KNclassifier = KNeighborsClassifier(n_neighbors=20)
KNclassifier.fit(X_train, y_train)

y_pred = KNclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
KNAcc = accuracy_score(y_pred,y_test)
print('K Neighbours accuracy is: {:.2f}%'.format(KNAcc*100))
```

```
              precision    recall  f1-score   support

       DrugY       0.65      0.73      0.69        30
       drugA       0.67      0.40      0.50         5
       drugB       0.00      0.00      0.00         3
       drugC       0.00      0.00      0.00         4
       drugX       0.71      0.83      0.77        18

    accuracy                           0.65        60
   macro avg       0.41      0.39      0.39        60
weighted avg       0.59      0.65      0.62        60

[[22  0  1  1  6]
 [ 3  2  0  0  0]
 [ 2  1  0  0  0]
 [ 4  0  0  0  0]
 [ 3  0  0  0 15]]
K Neighbours accuracy is: 65.00%
```

[30]:
```
scoreListknn = []
for i in range(1,30):
    KNclassifier = KNeighborsClassifier(n_neighbors = i)
    KNclassifier.fit(X_train, y_train)
    scoreListknn.append(KNclassifier.score(X_test, y_test))

plt.plot(range(1,30), scoreListknn)
plt.xticks(np.arange(1,30,1))
plt.xlabel("K value")
plt.ylabel("Score")
plt.show()
KNAccMax = max(scoreListknn)
print("KNN Acc Max {:.2f}%".format(KNAccMax*100))
```




```
KNN Acc Max 75.00%
```

## 7.3 Support Vector Machine (SVM)

[31]:
```
from sklearn.svm import SVC
SVCclassifier = SVC(kernel='linear', max_iter=251)
SVCclassifier.fit(X_train, y_train)

y_pred = SVCclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
SVCAcc = accuracy_score(y_pred,y_test)
print('SVC accuracy is: {:.2f}%'.format(SVCAcc*100))
```

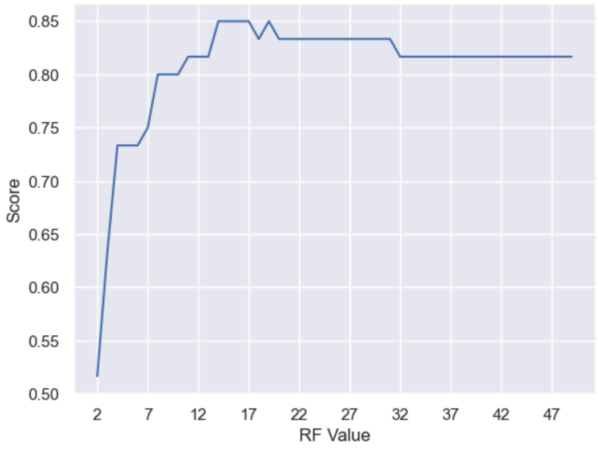
```
              precision    recall  f1-score   support

       DrugY       1.00      0.70      0.82        30
       drugA       0.71      1.00      0.83         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18

    accuracy                           0.85        60
   macro avg       0.79      0.94      0.84        60
weighted avg       0.89      0.85      0.85        60

[[21  2  1  2  4]
 [ 0  5  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0 18]]
SVC accuracy is: 85.00%
```

## 7.6 Random Forest

[32]:
```
from sklearn.ensemble import RandomForestClassifier

RFclassifier = RandomForestClassifier(max_leaf_nodes=30)
RFclassifier.fit(X_train, y_train)

y_pred = RFclassifier.predict(X_test)

print(classification_report(y_test, y_pred))
print(confusion_matrix(y_test, y_pred))

from sklearn.metrics import accuracy_score
RFAcc = accuracy_score(y_pred,y_test)
print('Random Forest accuracy is: {:.2f}%'.format(RFAcc*100))
```

```
              precision    recall  f1-score   support

       DrugY       0.95      0.70      0.81        30
       drugA       0.67      0.80      0.73         5
       drugB       0.75      1.00      0.86         3
       drugC       0.67      1.00      0.80         4
       drugX       0.82      1.00      0.90        18

    accuracy                           0.83        60
   macro avg       0.77      0.90      0.82        60
weighted avg       0.86      0.83      0.83        60

[[21  2  1  2  4]
 [ 1  4  0  0  0]
 [ 0  0  3  0  0]
 [ 0  0  0  4  0]
 [ 0  0  0  0 18]]
Random Forest accuracy is: 83.33%
```

[33]:
```
scoreListRF = []
for i in range(2,50):
    RFclassifier = RandomForestClassifier(n_estimators = 1000, random_state = 1, max_leaf_nodes=i)
    RFclassifier.fit(X_train, y_train)
    scoreListRF.append(RFclassifier.score(X_test, y_test))

plt.plot(range(2,50), scoreListRF)
plt.xticks(np.arange(2,50,5))
plt.xlabel("RF Value")
plt.ylabel("Score")
plt.show()
RFAccMax = max(scoreListRF)
print("RF Acc Max {:.2f}%".format(RFAccMax*100))
```




```
RF Acc Max 85.00%
```

## 8. Model Comparison

[34]:
```
compare = pd.DataFrame({'Model': ['Logistic Regression', 'K Neighbors'],
                        'Accuracy': [LRAcc*100,KNAcc*100]})
compare.sort_values(by='Accuracy', ascending=False)
```

[34]:

| | Model | Accuracy |
|---|---|---|
| **0** | Logistic Regression | 83.333333 |
| **1** | K Neighbors | 65.000000 |

# Experiment 9
# Cifar Dataset

## LOADING THE DATASET

[1]:
```
# pip install tensorflow
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')

import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
```

```
2024-03-31 14:28:55.425354: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register cuDNN factory: Attempting to r
egister factory for plugin cuDNN when one has already been registered
2024-03-31 14:28:55.425465: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register cuFFT factory: Attempting to re
gister factory for plugin cuFFT when one has already been registered
2024-03-31 14:28:55.584666: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to register cuBLAS factory: Attempting to
register factory for plugin cuBLAS when one has already been registered
```

[3]:
```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  2
```

[4]:
```
# Load the CIFAR-10 dataset
cifar10 = tf.keras.datasets.cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

# Exploring the dataset
print("Training set shape:", x_train.shape)
print("Test set shape:", x_test.shape)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
Training set shape: (50000, 32, 32, 3)
Test set shape: (10000, 32, 32, 3)
```

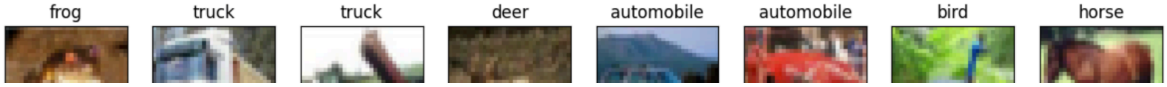
[5]:
```
print('Train Images Shape:      ', x_train.shape)
print('Train Labels Shape:      ', y_train.shape)
```

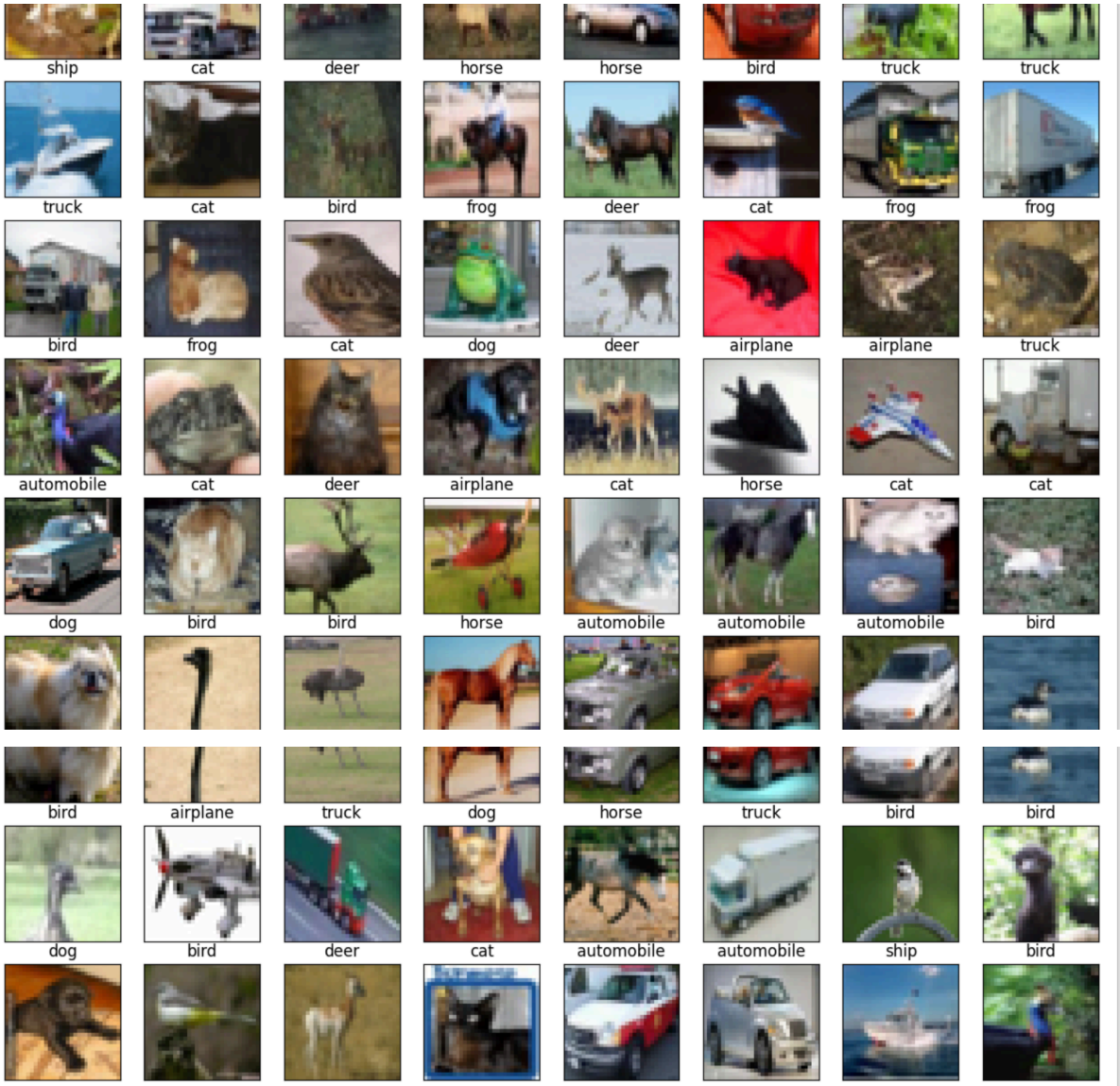
```
print('\nTest Images Shape:       ', x_test.shape)
print('Test Labels Shape:       ', y_test.shape)
```

```
Train Images Shape:       (50000, 32, 32, 3)
Train Labels Shape:       (50000, 1)

Test Images Shape:        (10000, 32, 32, 3)
Test Labels Shape:        (10000, 1)
```

### 2.2 | Explore

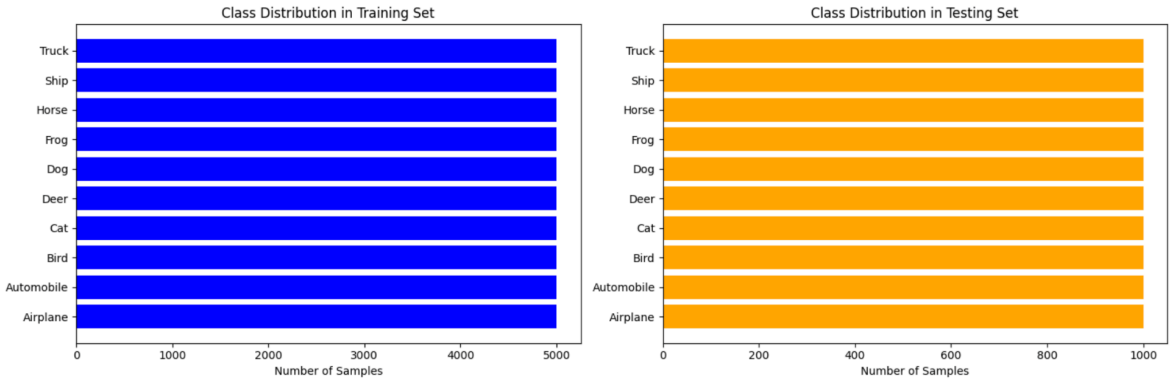
[6]:
```
# Exploring the CIFAR-10 dataset with and overview

# CIFAR-10 classes
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

# Create a new figure
plt.figure(figsize=(15,15))

# Loop over the first 25 images
for i in range(64):
    # Create a subplot for each image
    plt.subplot(8, 8, i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)

    # Display the image
    plt.imshow(x_train[i])

    # Set the label as the title
    plt.title(class_names[y_train[i][0]], fontsize=12)

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






[7]:
```
# Define class names
classes_name = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer', 'Dog', 'Frog', 'Horse', 'Ship', 'Truck']

# Get class distribution for training and testing sets
train_classes, train_counts = np.unique(y_train, return_counts=True)
test_classes, test_counts = np.unique(y_test, return_counts=True)

# Set figure size and create subplots
plt.figure(figsize=(15, 5))

# Plot class distribution for training set
plt.subplot(1, 2, 1)
plt.barh(classes_name, train_counts, color='blue')
plt.xlabel('Number of Samples')
plt.title('Class Distribution in Training Set')

# Plot class distribution for testing set
plt.subplot(1, 2, 2)
plt.barh(classes_name, test_counts, color='orange')
plt.xlabel('Number of Samples')
plt.title('Class Distribution in Testing Set')
```

```
# Adjust layout and display the plot
plt.tight_layout()
plt.show()
```




## NORMALIZATION ¶

[8]:
```
# Method 1: Normalization to [0,1] using Min-Max Scaling
x_train_min = np.min(x_train)
x_train_max = np.max(x_train)
x_train_normalized = (x_train - x_train_min) / (x_train_max - x_train_min)
```

[9]:
```
x_test_normalized = (x_test - x_train_min) / (x_train_max - x_train_min)
```

### 3.2 | Normalization to [-1,1] (Standardisation)

[10]:
```
# Method 2: Normalization to [-1,1] following a Normal Distribution
x_train_mean = np.mean(x_train)
x_train_std = np.std(x_train)
x_train_standardized = (x_train - x_train_mean) / x_train_std
```

[11]:
```
x_test_standardized = (x_test - x_train_mean) / x_train_std
```

## IMPLEMENTING A MULTI-LAYER PERCEPTRON FOR CIFAR-10 CLASSIFICATION

[12]:
```
# Flatten the images
x_train_flat = x_train_normalized.reshape(x_train_normalized.shape[0], -1)
x_test_flat = x_test_normalized.reshape(x_test_normalized.shape[0], -1)

# Display the shape of flattened images
print("Flattened Training Images Shape:", x_train_flat.shape)
print("Flattened Testing Images Shape:", x_test_flat.shape)
```

```
Flattened Training Images Shape: (50000, 3072)
Flattened Testing Images Shape: (10000, 3072)
```

### Model Architecture

[13]:
```
# Define the MLP model
model = tf.keras.models.Sequential(name="MLP_model")

# Add layers to the model
model.add(tf.keras.layers.Dense(512, activation='relu', input_shape=(3072,)))  # 3072 = 32x32x3
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

### Training

[14]:
```
from tensorflow.keras.optimizers import Adam

# Compile the model
optimizer = Adam()  # Use Adam optimizer
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
```

```
metrics=['accuracy'])

# Display the model summary
model.summary()
```

**Model: "MLP_model"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense (Dense) | (None, 512) | 1,573,376 |
| dense_1 (Dense) | (None, 10) | 5,130 |

**Total params:** 1,578,506 (6.02 MB)
**Trainable params:** 1,578,506 (6.02 MB)
**Non-trainable params:** 0 (0.00 B)

[15]:
```
# Train the model on the CIFAR-10 training set
history = model.fit(x_train_flat, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)  # Use 10% of training data as validation

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test_flat, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/10
 65/352 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1703 - loss: 3.1232
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
I0000 00:00:1711895366.107230     111 device_compiler.h:186] Compiled cluster using XLA!  This line is logged at most once for the lifetime o
f the process.
W0000 00:00:1711895366.122592     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.2588 - loss: 2.3075
W0000 00:00:1711895367.645605     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
W0000 00:00:1711895368.463756     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - accuracy: 0.2590 - loss: 2.3066 - val_accuracy: 0.3430 - val_loss: 1.8405
Epoch 2/10
 61/352 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.3600 - loss: 1.8030
W0000 00:00:1711895368.926628     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.3746 - loss: 1.7652 - val_accuracy: 0.3798 - val_loss: 1.7610
Epoch 3/10
```

## EVALUATION

[16]:
```
# Define the MLP model architectures
def create_mlp_model_1():
    model = tf.keras.models.Sequential([

        # Hidden layer with 128 neurons and ReLU activation
        tf.keras.layers.Dense(128, activation='relu', input_shape=(3072,)), # 3072 = 32x32x3

        # Output layer with softmax activation for multi-class classification
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    return model

def create_mlp_model_2():
    model = tf.keras.models.Sequential([

        # Hidden layer with 256 neurons and ReLU activation
        tf.keras.layers.Dense(256, activation='relu', input_shape=(3072,)), # 3072 = 32x32x3

        # Output layer with softmax activation for multi-class classification
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    return model

def create_mlp_model_3():
    model = tf.keras.models.Sequential([

        # First hidden layer with 256 neurons and ReLU activation
        tf.keras.layers.Dense(256, activation='relu', input_shape=(3072,)), # 3072 = 32x32x3

        # Second hidden layer with 128 neurons and ReLU activation
        tf.keras.layers.Dense(128, activation='relu'),

        # Output layer with softmax activation for multi-class classification
        tf.keras.layers.Dense(10, activation='softmax')
    ])
    return model
```

[17]:
```
# Compile and train the model
def compile_and_train_model(model):
    # Compile the model
    optimizer = Adam()  # Use Adam optimizer
    model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

    # Train the model on the CIFAR-10 training set
    history = model.fit(x_train_flat, y_train,
                        epochs=10,
                        batch_size=128,
                        validation_split=0.1)  # Use 10% of training data as validation

    # Evaluate the model on the test set
    test_loss, test_accuracy = model.evaluate(x_test_flat, y_test)
    print("Test Loss:", test_loss)
    print("Test Accuracy:", test_accuracy)
```

[18]:
```

# Create and train models
print("MLP Model with one hidden layer (128 neurons):")
mlp_model_1 = create_mlp_model_1()
compile_and_train_model(mlp_model_1)

print("\nMLP Model with one hidden layer (256 neurons):")
mlp_model_2 = create_mlp_model_2()
compile_and_train_model(mlp_model_2)

print("\nMLP Model with two hidden layers (256 and 128 neurons):")
mlp_model_3 = create_mlp_model_3()
compile_and_train_model(mlp_model_3)
```

```
MLP Model with one hidden layer (128 neurons):
Epoch 1/10
 78/352 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.1587 - loss: 2.4145
W0000 00:00:1711895383.044098     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.2433 - loss: 2.1226
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4885 - loss: 1.4639
Test Loss: 1.4678876399993896
Test Accuracy: 0.48339998722076416
```

[19]:
```
# Define the MLP model
model = tf.keras.models.Sequential(name="MLP_model_3layer")

# Add layers to the model
model.add(tf.keras.layers.Dense(512, activation='relu', input_shape=(3072,)))  # 3072 = 32x32x3
model.add(tf.keras.layers.Dense(256, activation='relu'))
model.add(tf.keras.layers.Dense(128, activation='relu'))
model.add(tf.keras.layers.Dense(10, activation='softmax'))
```

[20]:
```
# Compile the model
optimizer = Adam()  # Use Adam optimizer
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

[21]:
```
# Train the model on the CIFAR-10 training set
history = model.fit(x_train_flat, y_train,
                    epochs=20,
                    batch_size=256,
                    validation_split=0.1)  # Use 10% of training data as validation

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test_flat, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

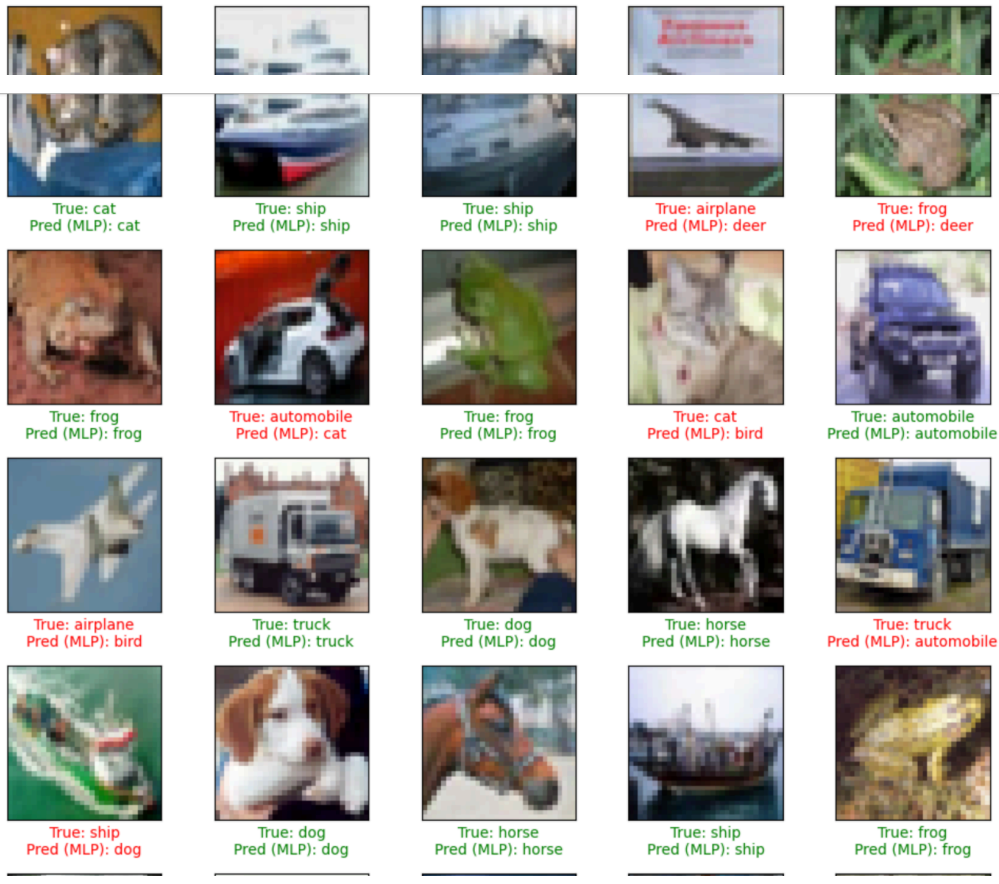
```
Epoch 1/20
 50/176 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.1785 - loss: 2.3376
W0000 00:00:1711895430.921345     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
176/176 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step - accuracy: 0.2445 - loss: 2.1107
W0000 00:00:1711895432.645227     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
W0000 00:00:1711895433.326721     113 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
176/176 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step - accuracy: 0.2448 - loss: 2.1098 - val_accuracy: 0.3342 - val_loss: 1.8283
Epoch 2/20
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.3726 - loss: 1.7579 - val_accuracy: 0.4050 - val_loss: 1.6716
Epoch 3/20
176/176 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4080 - loss: 1.6454 - val_accuracy: 0.4142 - val_loss: 1.6342
Epoch 4/20
```

```
Test Loss: 1.3978073596954346
Test Accuracy: 0.5188999772071838
W0000 00:00:1711895449.260978     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

## Visually checking the MLP model predictions

[22]:
```python
import numpy as np
import matplotlib.pyplot as plt

# Get predictions for the test set
predictions_mlp = model.predict(x_test_flat)
predicted_labels_mlp = np.argmax(predictions_mlp, axis=1)

# Plot sample images with predicted labels
plt.figure(figsize=(10, 10))
for i in range(25):  # Adjust as needed
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test[i])  # Plot original image
    true_label = class_names[y_test[i][0]]  # Get true label
    predicted_label_mlp = class_names[predicted_labels_mlp[i]]  # Get predicted label
    if true_label == predicted_label_mlp:
        color = 'green'  # Correct prediction
    else:
        color = 'red'  # Incorrect prediction
    plt.xlabel(f"True: {true_label}\nPred (MLP): {predicted_label_mlp}", color=color)
plt.tight_layout()  # Adjust subplot layout to prevent overlap
plt.show()
```

```
111/313 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
W0000 00:00:1711895449.931738     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
W0000 00:00:1711895450.583562     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```



# BUILDING A CONVOLUTIONAL NEURAL NETWORK FOR CIFAR-10

**Objective:** Develop a Convolutional Neural Network (CNN) to improve the classification performance on the CIFAR-10 dataset, and achieve >70% accuracy.

[23]:
```
# Convert to float32
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize the training set
x_train_min = x_train.min(axis=(0, 1, 2), keepdims=True)
x_train_max = x_train.max(axis=(0, 1, 2), keepdims=True)
x_train_normalized = (x_train - x_train_min) / (x_train_max - x_train_min)

# Normalize the test set using training set statistics
x_test_normalized = (x_test - x_train_min) / (x_train_max - x_train_min)

# Display the shapes of the normalized datasets
print("x_train_normalized shape:", x_train_normalized.shape)
print("x_test_normalized shape:", x_test_normalized.shape)
```

```
x_train_normalized shape: (50000, 32, 32, 3)
x_test_normalized shape: (10000, 32, 32, 3)
```

## CNN Architecture ¶

- Design a CNN that includes convolutional layers, activation functions, pooling layers, and fully connected layers.
- Detail your choice of kernel sizes, pooling sizes, and the architecture's depth.

[24]:
```
from tensorflow.keras import layers, models

# Define the CNN model
model = models.Sequential(name="CNN_model")

# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))

# Flatten layer to transition from convolutional to dense layers
model.add(layers.Flatten())

# Dense (fully connected) layers
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))  # Output layer with 10 classes
```

- The first layer has 32 filters, followed by a ReLU activation function, 'same' padding, and an input shape of (32, 32, 3) which corresponds to 32x32 RGB images.
- The subsequent layers continue to add convolutional layers with different numbers of filters and ReLU activations.
- MaxPooling layers with a pool size of (2, 2) are added after the convolutional layers to downsample the feature maps.
- A Flatten layer is added to transition from the convolutional layers to the dense layers. It flattens the input without affecting the batch size.
- We then add dense (fully connected) layers to the model. The first layer has 128 neurons with ReLU activation, and the output layer has 10 neurons (one for each class in CIFAR-10) with softmax activation, which is typical for multi-class classification problems.

## Training

- Compile your CNN with a suitable loss function and optimizer.
- Utilize techniques such as dropout and batch normalization to prevent overfitting and ensure more stable training.

[25]:
```
from tensorflow.keras import layers, models

# Define the CNN model
model = models.Sequential(name="CNN_model")

# Convolutional layers
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)))
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))  # Dropout layer to prevent overfitting
model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Dropout(0.25))  # Dropout layer
```

```
# Flatten layer to transition from convolutional to dense layers
model.add(layers.Flatten())

# Dense (fully connected) layers with batch normalization and dropout
model.add(layers.Dense(128, activation='relu'))
model.add(layers.BatchNormalization())  # Batch normalization layer
model.add(layers.Dropout(0.5))  # Dropout layer
model.add(layers.Dense(10, activation='softmax'))  # Output layer with 10 classes

# Compile the model with suitable loss function and optimizer
optimizer = Adam(learning_rate=0.0001)  # Use Adam optimizer
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Display the model summary
model.summary()
```

**Model: "CNN_model"**

| Layer (type) | Output Shape | Param # |
|---|---|---|
| conv2d_3 (Conv2D) | (None, 32, 32, 32) | 896 |
| conv2d_4 (Conv2D) | (None, 32, 32, 64) | 18,496 |
| max_pooling2d_2 (MaxPooling2D) | (None, 16, 16, 64) | 0 |
| dropout (Dropout) | (None, 16, 16, 64) | 0 |
| conv2d_5 (Conv2D) | (None, 16, 16, 64) | 36,928 |
| max_pooling2d_3 (MaxPooling2D) | (None, 8, 8, 64) | 0 |
| dropout_1 (Dropout) | (None, 8, 8, 64) | 0 |
| flatten_1 (Flatten) | (None, 4096) | 0 |
| dense_15 (Dense) | (None, 128) | 524,416 |
| batch_normalization (BatchNormalization) | (None, 128) | 512 |
| dropout_2 (Dropout) | (None, 128) | 0 |
| dense_16 (Dense) | (None, 10) | 1,290 |

**Total params:** 582,538 (2.22 MB)
**Trainable params:** 582,282 (2.22 MB)
**Non-trainable params:** 256 (1.00 KB)

## Evaluation and Comparison

- Report the accuracy of the model on the CIFAR-10 test set.
- Compare the performance of your CNN to the MLP model from Question 2, discussing why the CNN performs differently.

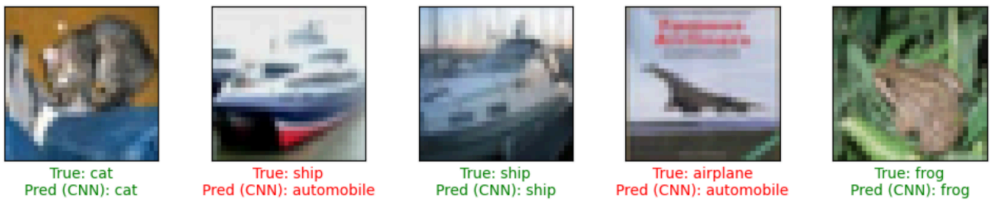
[26]:
```
# Fit the CNN model on the training set
history = model.fit(x_train_normalized, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_split=0.1)  # Use 10% of training data as validation

# Evaluate the CNN model on the test set
test_loss, test_accuracy = model.evaluate(x_test_normalized, y_test)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/10
 13/352 ━━━━━━━━━━━━━━━━━━━━ 4s 14ms/step - accuracy: 0.0962 - loss: 3.2341
W0000 00:00:1711895464.642245     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 0s 27ms/step - accuracy: 0.2070 - loss: 2.3559
W0000 00:00:1711895474.173790     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
W0000 00:00:1711895475.180702     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 21s 33ms/step - accuracy: 0.2072 - loss: 2.3550 - val_accuracy: 0.4000 - val_loss: 2.0001
Epoch 2/10
 13/352 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.3506 - loss: 1.7781
W0000 00:00:1711895476.166123     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.3790 - loss: 1.6971 - val_accuracy: 0.4772 - val_loss: 1.5103
Epoch 3/10
352/352 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 0.4504 - loss: 1.5079 - val_accuracy: 0.5234 - val_loss: 1.3609
Epoch 4/10
```

## Visually checking the model predictions (63.5% Accuracy)

[27]:
```
# Get predictions for the test set
predictions_cnn = model.predict(x_test_normalized)
predicted_labels_cnn = np.argmax(predictions_cnn, axis=1)

# Plot sample images with predicted labels
plt.figure(figsize=(10, 10))
for i in range(25):  # Adjust as needed
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(x_test_normalized[i])  # Plot original image
    true_label = class_names[y_test[i][0]]  # Get true label
    predicted_label = class_names[predicted_labels_cnn[i]]  # Get predicted label
    if true_label == predicted_label:
        color = 'green'  # Correct prediction
    else:
        color = 'red'  # Incorrect prediction
    plt.xlabel(f"True: {true_label}\nPred (CNN): {predicted_label}", color=color)
plt.tight_layout()  # Adjust subplot layout to prevent overlap
plt.show()
```

```
101/313 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
W0000 00:00:1711895509.796011     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
W0000 00:00:1711895510.633104     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```



## LEVRAGING A PRETRAINED MODEL FOR CIFAR-10

**Objective:** Use a pretrained model from TensorFlow's Keras applications or PyTorch's torchvision models as a feature extractor or for fine-tuning on the CIFAR-10 dataset, and achieve >90% accuracy on the test dataset of CIFAR-10. .

[28]:
```
# Load CIFAR-10 dataset
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

[29]:
```
from sklearn.model_selection import train_test_split

# Preprocess input images
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Preprocess input images using ResNet50 preprocessing
x_train = tf.keras.applications.resnet50.preprocess_input(x_train)
x_test = tf.keras.applications.resnet50.preprocess_input(x_test)
```

[30]:
```
from sklearn.model_selection import train_test_split

x_train,x_val,y_train,y_val=train_test_split(x_train,y_train,test_size=.1)
```

[31]:
```
from keras.utils import to_categorical

#One hot encode the labels.Since we have 10 classes we should expect the shape[1] of y_train,y_val and y_test to change from 1 to 10

y_train=to_categorical(y_train)
y_val=to_categorical(y_val)
y_test=to_categorical(y_test)
```

[32]:
```
#Print the dimensions of the datasets to check

print((x_train.shape,y_train.shape))
print((x_val.shape,y_val.shape))
print((x_test.shape,y_test.shape))
```

```
((45000, 32, 32, 3), (45000, 10))
((5000, 32, 32, 3), (5000, 10))
((10000, 32, 32, 3), (10000, 10))
```

[33]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Data Augmentation Function: Let's define an instance of the ImageDataGenerator class and set the parameters.
train_generator = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    shear_range=0.1,
    zoom_range=0.1,)

val_generator = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    shear_range=0.1,
    zoom_range=0.1,)

test_generator = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    shear_range=0.1,
    zoom_range=0.1,)
```

### Model Selection, Modification and Training Strategy

- Choose a pretrained model (e.g., ResNet, VGG16) and modify it for CIFAR-10 classification.
- Describe how you adapt the model for the 10 classes of CIFAR-10 (e.g., modifying the top layer, adjusting input size).
- Decide whether to freeze the weights of the pretrained layers and only train the top layer(s), or to fine-tune the entire network.
- Justify your choice based on the CIFAR-10 dataset's characteristics.

[34]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50  # Import ResNet50 model
from tensorflow.keras.layers import Dense, Flatten, GlobalAveragePooling2D, Dropout, BatchNormalization, UpSampling2D
from tensorflow.keras.models import Model
```

```
# Feature Extraction is performed by ResNet50 pretrained on imagenet weights.
# Input size is 224 x 224.
inputs = tf.keras.layers.Input(shape=(32, 32, 3))

# Upsample the input image to match the size expected by ResNet50
resized_inputs = UpSampling2D(size=(7, 7))(inputs)
```

[35]:
```
# Load the ResNet50 model with pretrained weights
resnet_model = ResNet50(
    include_top=False,
    weights='imagenet',
    input_tensor=resized_inputs  # Use resized input as input tensor
)

# Freeze all layers initially
for layer in resnet_model.layers:
    layer.trainable = False

# Unfreeze the last X layers of the ResNet50 model
NUM_LAYERS_TO_UNFREEZE = 30
for layer in resnet_model.layers[-NUM_LAYERS_TO_UNFREEZE:]:
    layer.trainable = True

# Global average pooling and classification layers
x = GlobalAveragePooling2D()(resnet_model.output)
x = Dense(1024, activation="relu")(x)
x = BatchNormalization()(x)
x = Dropout(0.3)(x)
x = Dense(512, activation="relu")(x)
x = BatchNormalization()(x)
x = Dropout(0.2)(x)
x = Dense(256, activation="relu")(x)
x = BatchNormalization()(x)
x = Dropout(0.2)(x)
classification_output = Dense(10, activation="softmax", name="classification")(x)

# Connect the feature extraction and "classifier" layers to build the model
ResNet_model = Model(inputs=inputs, outputs=classification_output, name="ResNet")

# Compile the model
optimizer = tf.keras.optimizers.Adam()  # Use Adam optimizer
```

```
#Specify the loss as categorical_crossentropy since the labels are 1 hot encoded.
#If labels are integer, use sparse categorical crossentropy as loss function.
ResNet_model.compile(optimizer=optimizer,
             loss='categorical_crossentropy',
             metrics=['accuracy'])

ResNet_model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_tf_kernels_notop.h
5
94765736/94765736 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

**Model: "ResNet"**

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_layer_7 (InputLayer) | (None, 32, 32, 3) | 0 | – |
| up_sampling2d (UpSampling2D) | (None, 224, 224, 3) | 0 | input_layer_7[0]… |
| conv1_pad (ZeroPadding2D) | (None, 230, 230, 3) | 0 | up_sampling2d[0]… |
| conv1_conv (Conv2D) | (None, 112, 112, 64) | 9,472 | conv1_pad[0][0] |

[36]:
```
# Count the number of layers
num_layers = len(ResNet_model.layers)
print("Number of layers in the model:", num_layers)
```

```
Number of layers in the model: 187
```

## Evaluation

- Test the model on the CIFAR-10 test set and report the accuracy.
- Compare the results with the MLP and CNN models from the previous questions, discussing the advantages and limitations of using pretrained models on a dataset like CIFAR-10.

[37]:
```
from keras.callbacks import ReduceLROnPlateau, EarlyStopping
```

[37]:
```
from keras.callbacks import ReduceLROnPlateau, EarlyStopping

# Add ReduceLROnPlateau callback
# Here, the learning rate will be reduced by half (factor=0.5) if no improvement in validation loss is observed for 1 epochs
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1, min_lr=0.00001)

# Add EarlyStopping callback
# Here, training will be stopped if no improvement in validation loss is observed for 5 epochs.
# The `restore_best_weights` parameter ensures that the model weights are reset to the values from the epoch
# with the best value of the monitored quantity (in this case, 'val_loss').
early_stopping = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True, verbose=1)


batch_size=64
epochs=15

history = ResNet_model.fit(train_generator.flow(x_train,y_train,batch_size=batch_size),
                                    epochs=epochs,
                                    steps_per_epoch=x_train.shape[0]//batch_size,
                                    validation_data=val_generator.flow(x_val,y_val,batch_size=batch_size),
                                    #validation_steps=250,
                                    callbacks=[reduce_lr, early_stopping],
                                    shuffle = True
                                   )
# Evaluate the model
test_loss, test_accuracy = ResNet_model.evaluate(test_generator.flow(x_test,y_test,batch_size=batch_size))
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
```

```
Epoch 1/15
W0000 00:00:1711895556.611916     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
150/703 ━━━━━━━━━━━━━━━━━━━━ 2:48 304ms/step - accuracy: 0.5475 - loss: 1.4262
W0000 00:00:1711895602.185296     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
703/703 ━━━━━━━━━━━━━━━━━━━━ 0s 219ms/step - accuracy: 0.6768 - loss: 0.9927
W0000 00:00:1711895715.069907     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
W0000 00:00:1711895728.566230     110 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
703/703 ━━━━━━━━━━━━━━━━━━━━ 213s 245ms/step - accuracy: 0.6769 - loss: 0.9923 - val_accuracy: 0.8082 - val_loss: 0.5523 - learning_rate: 0.0010
Epoch 2/15
703/703 ━━━━━━━━━━━━━━━━━━━━ 12s 17ms/step - accuracy: 0.8281 - loss: 0.4648 - val_accuracy: 0.8158 - val_loss: 0.5397 - learning_rate: 0.0010
Epoch 3/15
```

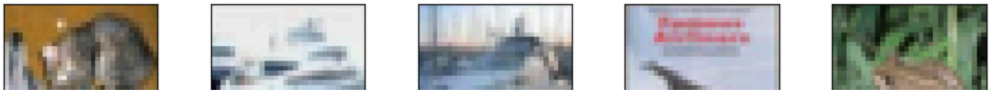
```
Test Loss: 0.27083495259284973
Test Accuracy: 0.9140999913215637
W0000 00:00:1711896885.253752     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

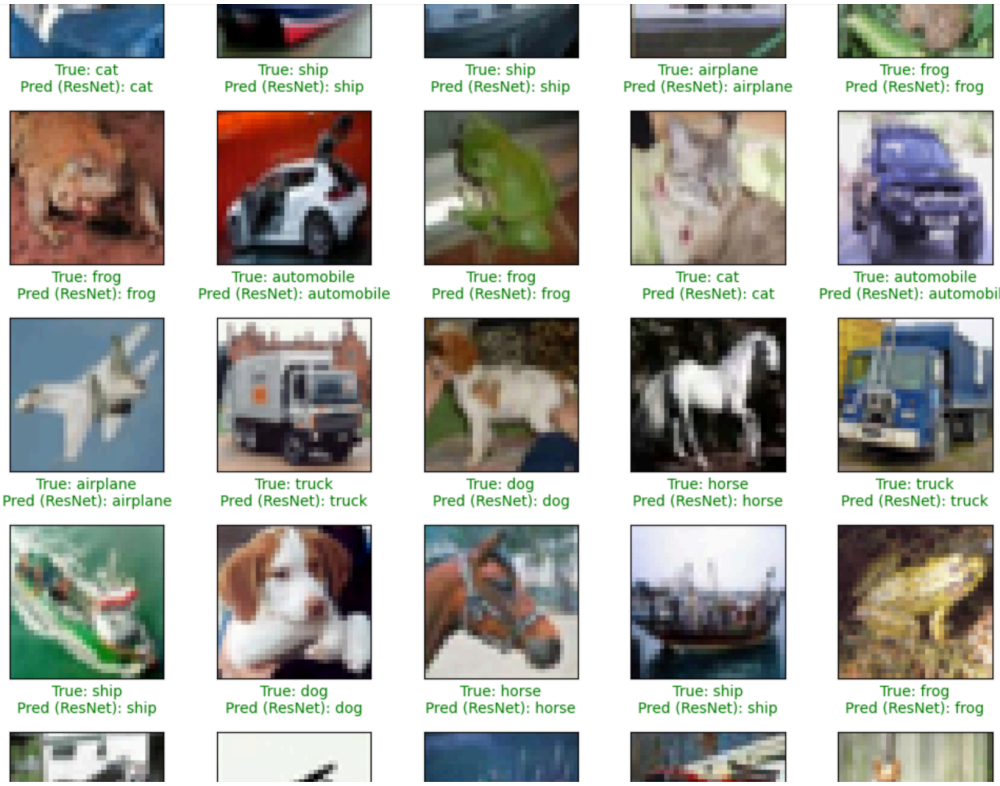
## Visually checking the model predictions (91.4% Accuracy)

[38]:
```
# Get predictions for the test set
predictions_resnet = ResNet_model.predict(x_test)
predicted_labels_resnet = np.argmax(predictions_resnet, axis=1)
```

```
  4/313 ━━━━━━━━━━━━━━━━━━━━ 19s 62ms/step
W0000 00:00:1711896895.673387     111 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
313/313 ━━━━━━━━━━━━━━━━━━━━ 37s 88ms/step
W0000 00:00:1711896923.093497     112 graph_launch.cc:671] Fallback to op-by-op mode because memset node breaks graph update
```

[39]:
```
# Plot sample images with predicted labels
plt.figure(figsize=(10, 10))
for i in range(25):  # Adjust as needed
    plt.subplot(5, 5, i + 1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    # Revert preprocessing for visualization by reloading CIFAR-10 dataset
    (x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
    plt.imshow(x_test[i])  #Plot original picture
    true_label = class_names[y_test[i][0]]  # Get true label
    predicted_label = class_names[predicted_labels_resnet[i]]  # Get predicted label
    if true_label == predicted_label:
        color = 'green'  # Correct prediction
    else:
        color = 'red'  # Incorrect prediction
    plt.xlabel(f"True: {true_label}\nPred (ResNet): {predicted_label}", color=color)
plt.tight_layout()  # Adjust subplot layout to prevent overlap
plt.show()
```




True: cat
Pred (ResNet): cat
True: ship
Pred (ResNet): ship
True: ship
Pred (ResNet): ship
True: airplane
Pred (ResNet): airplane
True: frog
Pred (ResNet): frog
True: frog
Pred (ResNet): frog
True: automobile
Pred (ResNet): automobile
True: frog
Pred (ResNet): frog
True: cat
Pred (ResNet): cat
True: automobile
Pred (ResNet): automobil
True: airplane
Pred (ResNet): airplane
True: truck
Pred (ResNet): truck
True: dog
Pred (ResNet): dog
True: horse
Pred (ResNet): horse
True: truck
Pred (ResNet): truck
True: ship
Pred (ResNet): ship
True: dog
Pred (ResNet): dog
True: horse
Pred (ResNet): horse
True: ship
Pred (ResNet): ship
True: frog
Pred (ResNet): frog