# Group B

# **Machine Learning**

**Experiment 1.**

**Predict the price of the Uber ride from a given pickup point to the agreed drop-off location.**

**Aim:**

1. **Pre-process the dataset.**
2. **Identify outliers.**
3. **Check the correlation.**
4. **Implement linear regression and random forest regression models.**
5. **Evaluate the models and compare their respective scores like R2, RMSE, etc.**

## Theory:

- **Dataset :**

  A dataset in machine learning is, quite simply, a collection of data pieces that can be treated by a computer as a single unit for analytic and prediction purposes.

- **Pre-Process:**

  Data pre-processing is a process of preparing the raw data and making it suitable for a machine learning model.

- **Outliers:**

  Sometimes a dataset can contain extreme values that are outside the range of what is expected and unlike the other data. These are called Outliers.

- **Correlation:**

  Correlation explains how one or more variables are related to each other.

- **Linear Regression:**

  Linear regression uses the relationship between the data-points to draw a straight line through all them.

- **Random Forest Regression:**

  Random forest is a supervised learning algorithm that uses an ensemble learning method for classification and regression. Random forest is a bagging technique and not a boosting technique. The trees in random forests run in parallel, meaning is no interaction between these trees while building the trees.

**Algorithm:**

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Finding the Outliers
- **Step 5:** Standardization
- **Step 6:** Splitting the Dataset
- **Step 7:** Random Forest Regression
- **Step 8:** Visualisation

**Database:** https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

**Program:**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pylab
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn import preprocessing

df = pd.read_csv('uber.csv')

df.info()
df.head()
df.describe()

df = df.drop(['Unnamed: 0', 'key'], axis=1)

df.isna().sum() df.dropna(axis=0,inplace=True)

df.dtypes

df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')df=

df.assign(
    second = df.pickup_datetime.dt.second,
    minute = df.pickup_datetime.dt.minute,hour
    = df.pickup_datetime.dt.hour, day=
    df.pickup_datetime.dt.day,
    month = df.pickup_datetime.dt.month,year =
    df.pickup_datetime.dt.year,
    dayofweek = df.pickup_datetime.dt.dayofweek
)
df = df.drop('pickup_datetime',axis=1)

incorrect_coordinates = df.loc[
    (df.pickup_latitude > 90) |(df.pickup_latitude < -90) |
    (df.dropoff_latitude > 90) |(df.dropoff_latitude < -90) |
    (df.pickup_longitude > 180) |(df.pickup_longitude < -180) |
    (df.dropoff_longitude > 90) |(df.dropoff_longitude < -90)
]

df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')

def distance_transform(longitude1, latitude1, longitude2, latitude2): long1, lati1,
    long2, lati2 = map(np.radians, [longitude1, latitude1,
longitude2, latitude2]) dist_long =
    long2 - long1dist_lati = lati2 -
```

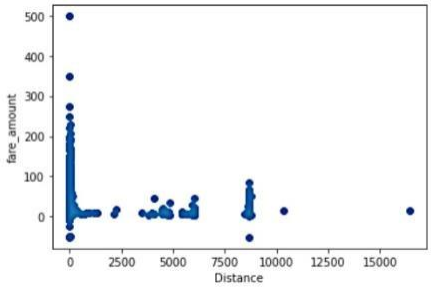
```
        lati1
        a = np.sin(dist_lati/2)**2 + np.cos(lati1) * np.cos(lati2) *
    np.sin(dist_long/2)**2
        c = 2 * np.arcsin(np.sqrt(a)) * 6371return
    c

    df['Distance'] =
        distance_transform(df['pickup_longit
        ude'], df['pickup_latitude'],
        df['dropoff_longitude'],
        df['dropoff_latitude']
    )

    plt.scatter(df['Distance'], df['fare_amount'])
    plt.xlabel("Distance") plt.ylabel("fare_amount")

    plt.figure(figsize=(20,12))
    sns.boxplot(data = df)

    df.drop(df[df['Distance'] >= 60].index, inplace = True) df.drop(df[df['fare_amount']
    <= 0].index, inplace = True)

    df.drop(df[(df['fare_amount']>100) & (df['Distance']<1)].index, inplace = True )
    df.drop(df[(df['fare_amount']<100) & (df['Distance']>100)].index, inplace=True )

    plt.scatter(df['Distance'], df['fare_amount'])
    plt.xlabel("Distance") plt.ylabel("fare_amount")

    corr = df.corr()

    corr.style.background_gradient(cmap='BuGn')

    X = df['Distance'].values.reshape(-1, 1)                    #Independent Variabley
    = df['fare_amount'].values.reshape(-1, 1)                   #Dependent Variable

    from sklearn.preprocessing import StandardScalerstd =
    StandardScaler()
    y_std = std.fit_transform(y)
    print(y_std)


  x_std = std.fit_transform(X)


    print(x_std)

    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(x_std, y_std, test_size=0.2,random_state=0)

    from sklearn.linear_model import LinearRegressionl_reg =
    LinearRegression()
    l_reg.fit(X_train, y_train)

    print("Training set score: {:.2f}".format(l_reg.score(X_train, y_train)))print("Test set
```

```
score: {:.7f}".format(l_reg.score(X_test, y_test)))

y_pred = l_reg.predict(X_test)

result = pd.DataFrame()
result[['Actual']] = y_test
result[['Predicted']] = y_pred

result.sample(10)

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred)) print('Mean
Absolute % Error:', metrics.mean_absolute_percentage_error(y_test,y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred)) print('Root
Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred)))

plt.subplot(2, 2, 1)
plt.scatter(X_train, y_train, color = 'red') plt.plot(X_train,
l_reg.predict(X_train), color ="blue")plt.title("Fare vs Distance
(Training Set)") plt.ylabel("fare_amount")
plt.xlabel("Distance")

plt.subplot(2, 2, 2)
plt.scatter(X_test, y_test, color = 'red') plt.plot(X_train,
l_reg.predict(X_train), color ="blue")plt.ylabel("fare_amount")
plt.xlabel("Distance")
plt.title("Fare vs Distance (Test Set)")

plt.tight_layout()
plt.show()

cols = ['Model', 'RMSE', 'R-Squared']

# create a empty dataframe of the columns: specifies the columns to be selectedresult_tabulation =
pd.DataFrame(columns = cols)

# compile the required information
linreg_metrics = pd.DataFrame([[
        "Linear Regresion model",
        np.sqrt(metrics.mean_squared_error(y_test, y_pred)),
        np.sqrt(metrics.r2_score(y_test, y_pred))
]], columns = cols)
result_tabulation = pd.concat([result_tabulation, linreg_metrics],
ignore_index=True)


result_tabulation


rf_reg = RandomForestRegressor(n_estimators=100, random_state=10)# fit

the regressor with training dataset
rf_reg.fit(X_train, y_train)

# predict the values on test dataset using predict()y_pred_RF
= rf_reg.predict(X_test)
```

```
result = pd.DataFrame()
result[['Actual']] = y_test
result['Predicted'] = y_pred_RF

result.sample(10)


print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred_RF))
print('Mean Absolute % Error:', metrics.mean_absolute_percentage_error(y_test,
y_pred_RF))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred_RF)) print('Root Mean
Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,y_pred_RF)))
print('R Squared (R²):', np.sqrt(metrics.r2_score(y_test, y_pred_RF)))


# Build scatterplot
plt.scatter(X_test, y_test, c = 'b', alpha = 0.5, marker = '.', label = 'Real')plt.scatter(X_test,
y_pred_RF, c = 'r', alpha = 0.5, marker = '.', label = 'Predicted')
plt.xlabel('Carat')
plt.ylabel('Price')
plt.grid(color = '#D3D3D3', linestyle = 'solid')plt.legend(loc
= 'lower right')


plt.tight_layout()
plt.show()


# compile the required information
random_forest_metrics = pd.DataFrame([[
        "Random Forest Regressor model",
        np.sqrt(metrics.mean_squared_error(y_test, y_pred_RF)),
        np.sqrt(metrics.r2_score(y_test, y_pred_RF))
]], columns = cols)


result_tabulation = pd.concat([result_tabulation, random_forest_metrics],ignore_index=True)


result_tabulation
```

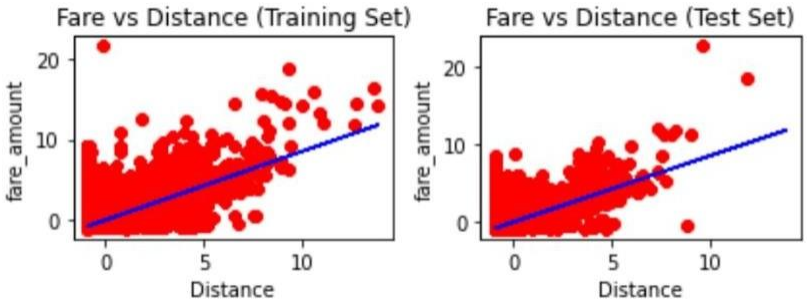
**Output:**

**Outliers:**



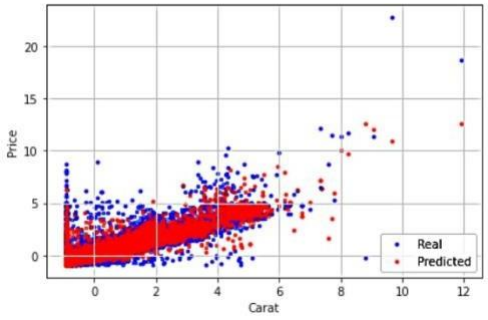
**Linear Regression:**

**Mean Absolute Error:**
**0.26621298757938955 Mean Absolute %**
**Error: 1.983074763340738 Mean**
**Squared Error: 0.2705243510778542**
**Root Mean Squared Error:**
**0.5201195546005305R Squared ($R^2$):**
**0.8567653080822022**




**Random Forests Regression:**

**Mean Absolute Error:**
**0.3077087698385678 Mean Absolute %**
**Error: 2.161623761570947 Mean**
**Squared Error: 0.33297733033643484**




**Root Mean Squared Error:**
**0.5770418791876677R Squared ($R^2$):**
**0.8201518783882692**

**Conclusion:**

**Hence, we here we studied that what is the dataset, correlation, outliers, linear regression and random forest regression with programmatic visualisation.**

**Experiment 2.**

**Classify the email using the binary classification method.**

**Aim:**

**Analyze their performance.**

a) **Normal State – Not Spam**

b) **Abnormal State – Spam.**

**Use K-Nearest Neighbors and Support Vector Machine for classification.**

**Theory:**

- **Binary Classification:**
  In machine learning, binary classification is a supervised learning algorithm that categorizes new observations into one of two classes.
- **K-Nearest Neighbors:**
  The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another.
- **Support Vector Machine:**
  Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

**Algorithm:**

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Separating the features and the labels
- **Step 5:** Splitting the Dataset
- **Step 6: Machine Learning models**
  - The following 5 models are used:
    - K-Nearest Neighbors
    - Linear SVM
    - Polynomial SVM
    - RBF SVM
    - Sigmoid SVM
- **Step 7:** Fit and predict on each model

**Database:** https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

**Program:**

```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics, preprocessing

df = pd.read_csv('emails.csv')

df.info()
df.head()

df.drop(columns=['Email No.'], inplace=True)
df.isna().sum()
df.describe()

X=df.iloc[:, :df.shape[1]-1]                    #Independent Variables
y=df.iloc[:, -1]                                #Dependent Variable
X.shape, y.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,random_state=8)

models = {
    "K-Nearest Neighbors": KNeighborsClassifier(n_neighbors=2), "Linear
    SVM":LinearSVC(random_state=8, max_iter=900000), "Polynomical
    SVM":SVC(kernel="poly", degree=2, random_state=8),"RBF
    SVM":SVC(kernel="rbf", random_state=8),
    "Sigmoid SVM":SVC(kernel="sigmoid", random_state=8)
}

for name, model in models.items(): y_pred=model.fit(X_train,
    y_train).predict(X_test)
    print(f"Accuracy for {name} model \t: {metrics.accuracy_score(y_test,y_pred)}")
```

**Output:**

```
Accuracy for K-Nearest Neighbors model : 0.8878865979381443
Accuracy for Linear SVM model          : 0.9755154639175257
Accuracy for Polynomical SVM model :  0.7615979381443299
Accuracy for RBF SVM model             : 0.8182989690721649
Accuracy for Sigmoid SVM model         : 0.6237113402061856
```

**Conclusion:**

**Hence, we studied here, Classification of email using K-Nearest Neighbors, Linear SVM, Polynomial SVM, RBF SVM, Sigmoid SVM, for Analyze their performance..**

**Experiment 3.**

**Given a bank customer, build a neural network-based classifier that can determine whether they will leave or not in the next 6 months.**

**Aim:**

1. **Read the dataset.**
2. **Distinguish the feature and target set and divide the data set into training and test sets.**
3. **Normalize the train and test data.**
4. **Initialize and build the model. Identify the points of improvement and implement the same.**
5. **Print the accuracy score and confusion matrix (5 points).**

**Theory:**

- **Feature:**

  Features are nothing but the independent variables in machine learning models. What is required to be learned in any specific machine learning problem is a set of these features (independent variables), coefficients of these features, and parameters for coming up with appropriate functions or models (also termed hyper parameters).

- **Training Set:**

  Training data is also known as training dataset, learning set, and training set. It's an essential component of every machine learning model and helps them make accurate predictions or perform a desired task.

- **Test Set:**

  The test set is a set of observations used to evaluate the performance of the model using some performance metric. It is important that no observations from the training set are included in the test set. If the test set does contain examples from the training set, it will be difficult to assess whether the algorithm has learned to generalize from the training set or has simply memorized it.

- **Normalisation:**

  Normalization is a scaling technique in Machine Learning applied during data preparation to change the values of numeric columns in the dataset to use a common scale. It is not necessary for all datasets in a model. It is required only when features of machine learning models have different ranges.

- **Model:**

  A “model” in machine learning is the output of a machine learning algorithm run on data.

  A model represents what was learned by a machine learning algorithm.

**Algorithm:**

- **Step 1:** Importing required libraries
- **Step 2:** Read the Dataset
- **Step 3:** Cleaning
- **Step 4:** Separating the features and the labels
- **Step 5:** Splitting the Dataset
- **Step 6:** Initialize & build the model
- **Step 7:** Predict the results using 0.5 as a threshold
- **Step 8:** Print the Accuracy score and confusion matrix

**Database:** https://www.kaggle.com/barelydedicated/bank-customer-churn-modeling

**Program:**

```
import pandas as pd, numpy as np, seaborn as sns, matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder, StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics, preprocessing

df = pd.read_csv('churn_modelling.csv')
df.info()
df.head()

df.drop(columns=['RowNumber', 'CustomerId', 'Surname'], inplace=True)
df.isna().sum()
df.describe()

X=df.iloc[:, :df.shape[1]-1].values                    #Independent Variables
y=df.iloc[:, -1].values                                #Dependent Variable
X.shape, y.shape

print(X[:8,1], '... will now become: ')

label_X_country_encoder = LabelEncoder()
X[:,1] = label_X_country_encoder.fit_transform(X[:,1])
print(X[:8,1])

print(X[:6,2], '... will now become: ')

label_X_gender_encoder = LabelEncoder()
X[:,2] = label_X_gender_encoder.fit_transform(X[:,2])

print(X[:6,2])

transform = ColumnTransformer([("countries", OneHotEncoder(), [1])],
remainder="passthrough") # 1 is the country column
X = transform.fit_transform(X)

X = X[:,1:]
X.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=0)
```

```
sc=StandardScaler()
X_train[:,np.array([2,4,5,6,7,10])] =
sc.fit_transform(X_train[:,np.array([2,4,5,6,7,10])])
X_test[:,np.array([2,4,5,6,7,10])] =
sc.transform(X_test[:,np.array([2,4,5,6,7,10])])

sc=StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
X_train

from tensorflow.keras.models import Sequential#

Initializing the ANN
classifier = Sequential()

from tensorflow.keras.layers import Dense

# The amount of nodes (dimensions) in hidden layer should be the average of inputand output
layers, in this case 6.
# This adds the input layer (by specifying input dimension) AND the first hiddenlayer (units)
classifier.add(Dense(activation = 'relu', input_dim = 11, units=256,kernel_initializer='uniform'))

# Adding the hidden layer classifier.add(Dense(activation =
'relu', units=512,kernel_initializer='uniform'))
classifier.add(Dense(activation = 'relu', units=256,
kernel_initializer='uniform')) classifier.add(Dense(activation
= 'relu', units=128,kernel_initializer='uniform'))

# Adding the output layer
# Notice that we do not need to specify input dim.
# we have an output of 1 node, which is the the desired dimensions of our output(stay with the bank
or not)
# We use the sigmoid because we want probability outcomes
classifier.add(Dense(activation = 'sigmoid', units=1, kernel_initializer='uniform'))

# Create optimizer with default learning rate#
sgd_optimizer  =  tf.keras.optimizers.SGD() #
Compile the model



classifier.compile(optimizer='adam', loss='binary_crossentropy',metrics=['accuracy'])

classifier.summary()

classifier.fit(
      X_train, y_train,
      validation_data=(X_test,y_test),epochs=20,
      batch_size=32
)

y_pred = classifier.predict(X_test)
```

```
# To use the confusion Matrix, we need to convert the probabilities that acustomer will
leave the bank into the form true or false.
# So we will use the cutoff value 0.5 to indicate whether they are likely to exitor not.
y_pred = (y_pred > 0.5)

from sklearn.metrics import confusion_matrix,classification_reportcm1 =

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

accuracy_model1 = ((cm1[0][0]+cm1[1][1])*100)/(cm1[0][0]+cm1[1][1]+cm1[0][1]+cm1[1][0])
print (accuracy_model1, '% of testing data was classified correctly')
```

**Output:**

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.88 | 0.96 | 0.92 | 1595 |
| 1 | 0.74 | 0.49 | 0.59 | 405 |
| accuracy | | | 0.86 | 2000 |
| macro avg | 0.81 | 0.72 | 0.75 | 2000 |
| weighted avg | 0.85 | 0.86 | 0.85 | 2000 |

**86.15 % of testing data was classified correctly**

**Conclusion:**

**Hence, we studied that, how is the training & test data sets are. What is model and how to builtit. Classified testing data set.**

.

**Experiment 4.**

**Implement Gradient Descent Algorithm to find the local minima of a function.**

**Aim:**

**Find the local minima of the function $y=(x+3)^2$ starting from the point x=2.**

**Theory:**

**Gradient Descent is an optimization algorithm used for minimizing the cost function in various machine learning algorithms. It is basically used for updating the parameters of the learning model.**

**Types of gradient Descent:**

- **Batch Gradient Descent:**
  This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.

- **Stochastic Gradient Descent:**
  This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.

- **Mini Batch gradient descent:**
  This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here b examples where b<m are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

## Algorithm:

- **Gradient Descent:**

**Algorithm 1:** Pseudo-code for GD

**Function** `GD`**:**
  Set epsilon as the limit of convergence
  **for** $j = 1$ *and* $j = 0$ **do**
    **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
      $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{m} \sum_{i=1}^{m} (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
    **end**
  **end**

- **Stochastic Gradient Descent**

**Algorithm 2:** Pseudo-code for SGD

**Function** `SGD`**:**
  Set epsilon as the limit of convergence
  **for** $i = 1, \ldots, m$ **do**
    **for** $j = 0, \ldots, n$ **do**
      **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
        $\omega_{j+1} := \omega_j - \alpha \cdot (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
      **end**
    **end**
  **end**

- **Mini Batch Gradient Descent**

**Algorithm 3:** Pseudo-code for Mini Batch Gradient Desecent

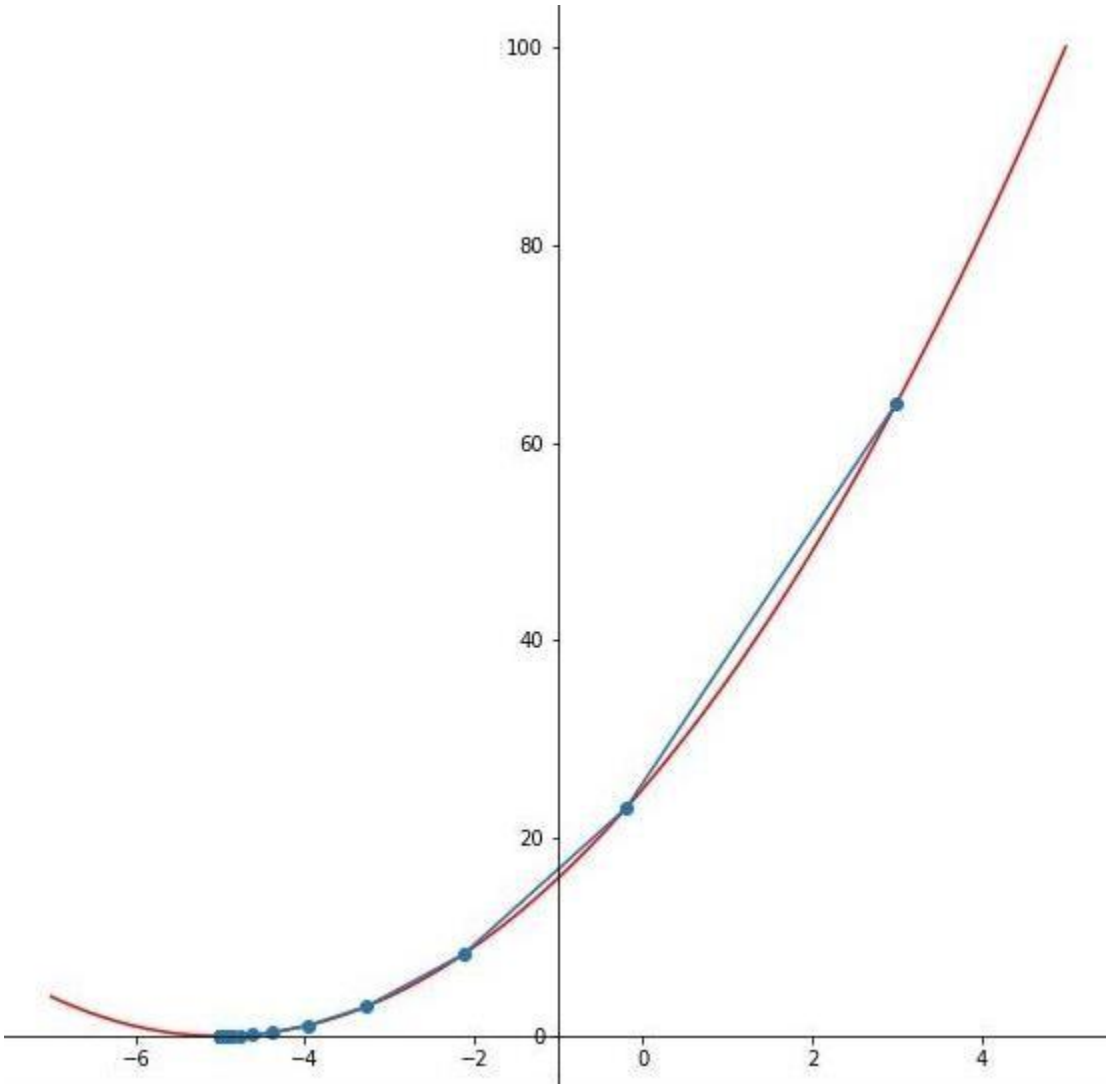
**Function** `SGD`**:**
  Set epsilon as the limit of convergence
  **for** $i = 1, \ldots, b$ **do**
    **for** $j = 0, \ldots, n$ **do**
      **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
        $\omega_{j+1} := \omega_j - \alpha \cdot \frac{1}{b} \sum_{k=i}^{i+b-1} (h_\omega(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$;
      **end**
    **end**
  **end**

**Program:**

```
from sympy import Symbol, lambdify
import matplotlib.pyplot as plt import
numpy as np

x = Symbol('x')

def gradient_descent(
    function, start, learn_rate, n_iter=10000, tolerance=1e-06, step_size=1 ):gradient =
    lambdify(x, function.diff(x))
    function = lambdify(x, function)points =
    [start]
    iters = 0                                   #iteration counter

    while step_size > tolerance and iters < n_iter:
        prev_x = start                          #Store current x value in prev_xstart
        = start - learn_rate * gradient(prev_x) #Grad descent step_size = abs(start
        - prev_x) #Change in x
        iters = iters+1                         #iteration count
        points.append(start)
    print("The local minimum occurs at", start)

    # Create plotting array
    x_ = np.linspace(-7,5,100)y =
    function(x_)

    # setting the axes at the centre fig =
    plt.figure(figsize = (10, 10))
    ax = fig.add_subplot(1, 1, 1) ax.spines['left'].set_position('center')
    ax.spines['bottom'].set_position('zero')
    ax.spines['right'].set_color('none') ax.spines['top'].set_color('none')
    ax.xaxis.set_ticks_position('bottom') ax.yaxis.set_ticks_position('left')

    # plot the function
    plt.plot(x_,y, 'r')
    plt.plot(points, function(np.array(points)), '-o')

    # show the plot
    plt.show()

function=(x+5)**2

gradient_descent(
     function=function, start=3.0, learn_rate=0.2, n_iter=50
)
```

**Output:**

**The local minimum occurs at -4.999998938845185**



**Conclusion:**

**Hence, we learn Gradient Descent Algorithm and it's types. And program for calculating local minima.**

.

**Experiment 5.**

**Implement K-Nearest Neighbors algorithm on diabetes.csv dataset.**

**Aim:**

**Compute confusion matrix, accuracy, error rate, precision and recall on the given dataset.**

**Theory:**

- **Nearest Neighbors algorithm:**

  - K-Nearest Neighbour is one of the simplest Machine Learning algorithms based on Supervised Learning technique.

  - K-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.

  - K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.

  - K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.

  - K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.

  - It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.

  - KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

**Algorithm:**

- **Step 1:** Importing libraries
- **Step 2:** Loading the Dataset
- **Step 3:** Cleaning
- **Step 4:** Visualization
- **Step 5:** Separating the features and the labels
- **Step 6:** Splitting the Dataset
- **Step 7:** Machine Learning model
- **Step 8:** Hyper parameter tuning

**Database: https://www.kaggle.com/datasets/abdallamahgoub/diabetes**

**Program:**

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix, classification_report,accuracy_score
from sklearn import preprocessing

df = pd.read_csv('diabetes.csv')
df.info()
df.head()

df.corr().style.background_gradient(cmap='BuGn')
df.isna().sum()
df.describe()

hist = df.hist(figsize=(20,16))

X=df.iloc[:, :df.shape[1]-1]                    #Independent Variables
y=df.iloc[:, -1]                                #Dependent Variable
X.shape, y.shape

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=8)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)


def knn(X_train, X_test, y_train, y_test, neighbors, power): model =
    KNeighborsClassifier(n_neighbors=neighbors, p=power)# Fit and
    predict on model
    # Model is trained using the train set and predictions are made based on thetest set.
Accuracy scores are calculated for the model.
    y_pred=model.fit(X_train, y_train).predict(X_test)
    print(f"Accuracy for K-Nearest Neighbors model \t: {accuracy_score(y_test,y_pred)}")

    cm = confusion_matrix(y_test, y_pred)
    print(f"'Confusion matrix :\n
| Positive Prediction\t| Negative Prediction
```

```
-------------+----------------------+--------------------
Positive Class | True Positive (TP) {cm[0, 0]}\t| False Negative (FN) {cm[0, 1]}
-------------+----------------------+--------------------
Negative Class|False Positive(FP){cm[1,0]}\t| True Negative (TN) {cm[1,1]}\n''')cr =
    classification_report(y_test, y_pred)
    print('Classification report : \n', cr)

param_grid = {
    'n_neighbors': range(1, 51),
    'p': range(1, 4)
}
grid=GridSearchCV(estimator=KNeighborsClassifier(),param_grid=param_grid,cv=5)
grid.fit(X_train, y_train)
grid.best_estimator_, grid.best_params_, grid.best_score_

knn(X_train, X_test, y_train, y_test, grid.best_params_['n_neighbors'],grid.best_params_['p'])
```

**Output:**

```
Accuracy for K-Nearest Neighbors model  : 0.7987012987012987
Confusion matrix :
    | Positive Prediction              | Negative Prediction
    -------------+----------------------+--------------------
    Positive Class | True Positive (TP) 91              | False Negative (FN) 11
    -------------+----------------------+--------------------
    Negative Class | False Positive (FP) 20             | True Negative (TN) 32
Classification report :
                  precision    recall  f1-score   support

               0       0.82      0.89      0.85       102
               1       0.74      0.62      0.67        52

        accuracy                           0.80       154
       macro avg       0.78      0.75      0.76       154
    weighted avg       0.79      0.80      0.79       154
```

**Conclusion:**

**Hence, here we learned that, confusion matrix, accuracy, error rate, precision and recall, by performing it in program.**

.

**Experiment 6.**

**Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset.**

**Aim:**

**Determine the number of clusters using the elbow method.**

**Theory:**

- **K-Means Clustering:**

  The most common clustering covered in machine learning for beginners is K-Means. The first step is to create c new observations among our unlabelled data and locate them randomly, called centroids. The number of centroids represents the number of output classes. The first step of the iterative process for each centroid is to find the nearest point (in terms of Euclidean distance) and assign them to its category. Next, for each category, the average of all the points attributed to that class is computed. The output is the new centroid of the class. With every iteration, the observations can be redirected to another centroid. After several reiterations, the centroid's change in location is less critical as initial random centroids converge with real ones—the process ends when there is no change in centroids' position. Many methods can be employed for the task, but a common one is 'elbow method'. A low level of variation is needed within the clusters measured by the within-cluster sum of squares. The number of centroids and observations are inversely proportional. Thus, setting the highest possible number of centroids would be inconsistent.

- **Hierarchical Clustering:**

  Two techniques are used by this algorithm- Agglomerative and Divisive. In HC, the number of clusters K can be set precisely like in K-means, and n is the number of data points such that n>K. The agglomerative HC starts from n clusters and aggregates data until K clusters are obtained. The divisive starts from only one cluster and then splits depending on similarities until K clusters are obtained. The similarity here is the distance among points, which can be computed in many ways, and it is the crucial element of discrimination. It can be computed with different approaches:

  **Min:** Given two clusters C1 and C2 such that point a belongs to C1 and b to C2. The similarity between them is equal to the minimum of distance

  **Max**: The similarity between points a and b is equal to the maximum of distance

  **Average:** All the pairs of points are taken, and their similarities are computed. Then the average of similarities is the similarity between C1 and C2.

**Algorithm:**

- **Step 1:** Importing Libraries
- **Step 2:** Reading the dataset
- **Step 3:** Creating Datalist
- **Step 4:** Applying K-mean algorithm and findout elbow points
- **Step 5:** Creating Clusters
- **Step 6:** Plotting the elbow plot

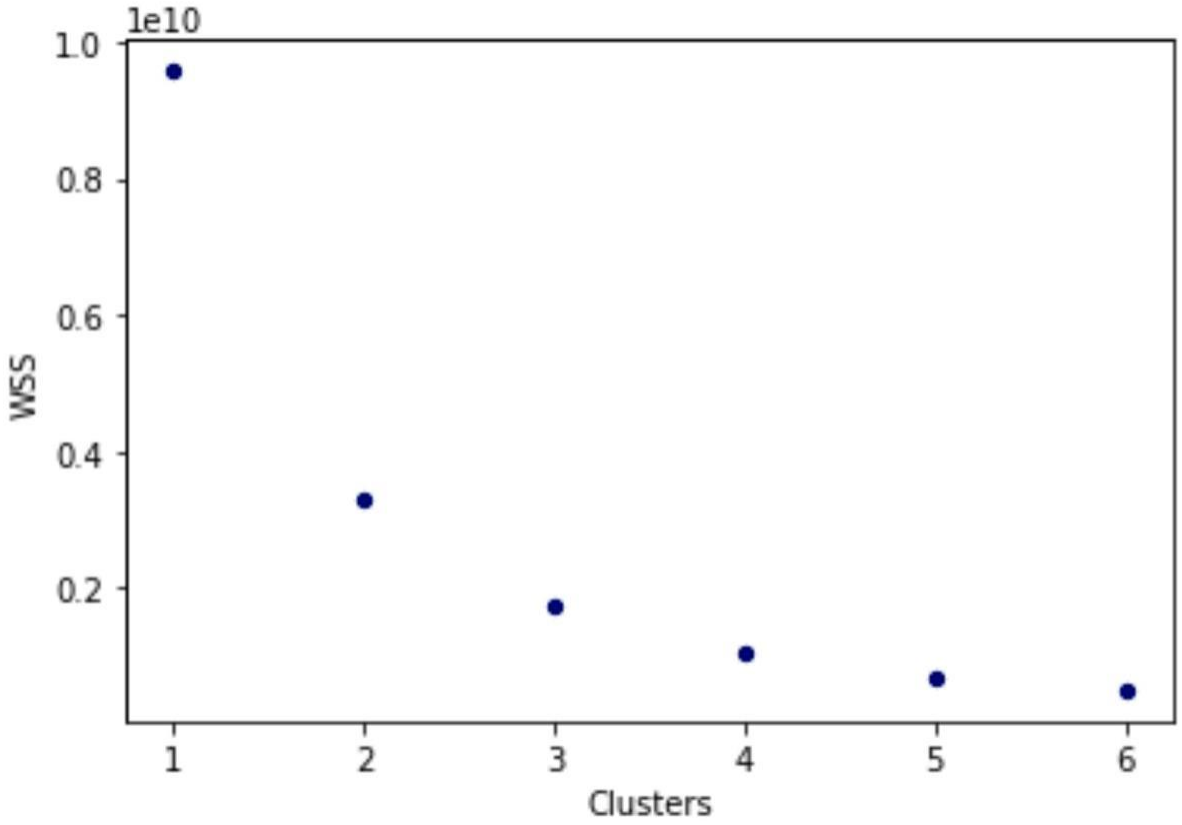
**Database: https://www.kaggle.com/datasets/kyanyoga/sample**

**Program:**

```
import pandas as pd import
numpy as np import seaborn
as sns
import matplotlib.pyplot as plt import
sklearn.cluster as cluster

%matplotlib inline

df = pd.read_csv('sales_data_sample.csv', encoding='Latin-1')df.head()

list = ["PRICEEACH", "SALES"]
data = df[list]
data.head()

K=range(1, 7)wss
= []
for k in K:
    kmeans=cluster.KMeans(n_clusters=k,init="k-means++")kmeans=kmeans.fit(data)
    wss_iter = kmeans.inertia_
    wss.append(wss_iter)

mycenters = pd.DataFrame({'Clusters' : K, 'WSS' : wss})mycenters

sns.scatterplot(x = 'Clusters', y = 'WSS', data = mycenters)
```

**Output:**

| Clusters | | WSS |
|---|---|---|
| **0** | **1** | **9.574691e+09** |
| **1** | **2** | **3.322310e+09** |
| **2** | **3** | **1.729502e+09** |
| **3** | **4** | **1.040298e+09** |
| **4** | **5** | **7.126630e+08** |
| **5** | **6** | **5.119084e+08** |



**Conclusion:**

**Hence, we studied about K-Means clustering & hierarchical clustering. And also performed program on sales data.**

**Experiment 7. Mini Project.**

**Aim:**

**Build a machine Learning model that predicts the type of people who survived the Titanic shipwreck using Passenger data (i.e. name, age, gender, socio-economic class, etc.).**

**Theory:**

- **Machine Learning:**

  Machine learning is a field of computer science that aims to teach computers how to learn and act without being explicitly programmed. More specifically, machine learning is an approach to data analysis that involves building and adapting models, which allow programs to "learn" through experience.

- **Model in Machine Learning:**

  A machine learning model is a program that can find patterns or make decisions from a previously unseen dataset. For example, in natural language processing, machine learning models can parse and correctly recognize the intent behind previously unheard sentences or combinations of words.

- **Prediction:**

  Prediction in machine learning refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome.

**Algorithm:**

- **Step 1:** Importing Required libraries
- **Step 2:** Reading Training & Test Datasets.
- **Step 3:** Cleaning
- **Step 4:** Splitting Datasets
- **Step 5:** Training the Model
- **Step 6:** Testing the Model
- **Step 7:** Use maodel for Prediction.

**Database:** **https://www.kaggle.com/competitions/titanic/data**

**Program:**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import cross_val_scorefrom
sklearn.ensemble import RandomForestClassifier

train = pd.read_csv('train.csv')test =
pd.read_csv('test.csv')

train.info()
test.info()

train.head()
train.head()

train.corr().style.background_gradient(cmap='BuGn')

train.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
test.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1, inplace=True)

train.isna().sum()
test.isna().sum()

train['Embarked'] = train.Embarked.fillna(train.Embarked.dropna().max())
test['Fare'] = test.Fare.fillna(test.Fare.dropna().mean())

# we will guess the age from Pclass and Sex:

guess_ages = np.zeros((2,3))

combine = [train , test]

# Converting Sex categories (male and female) to 0 and 1:
for ds in combine:
    ds['Sex'] = ds['Sex'].map( {'female': 1, 'male': 0} ).astype(int)

# Filling missed age feature:
for ds in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = ds[(ds['Sex'] == i) & \
                                    (ds['Pclass'] == j+1)]['Age'].dropna()
            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    for i in range(0, 2):
```

```
            for j in range(0, 3):
                ds.loc[ (ds.Age.isnull()) & (ds.Sex == i) & (ds.Pclass == j+1),\'Age'] =
                        guess_ages[i,j]

    ds['Age'] = ds['Age'].astype(int)


train.head()
train.describe()

X_train = pd.get_dummies(train.drop(['Survived'], axis=1))X_test   =
pd.get_dummies(test)
y_train = train['Survived']

def print_scores(model, X_train, Y_train, predictions, cv_splites=10):print("The
    mean accuracy score of the train data is %.5f" %
model.score(X_train, Y_train))
    CV_scores = cross_val_score(model, X_train, Y_train, cv=cv_splites)
    print("The individual cross-validation scores are: \n",CV_scores) print("The
    minimum cross-validation score is %.3f" % min(CV_scores))print("The
    maximum cross-validation score is %.3f" % max(CV_scores))print("The mean
    cross-validation                    score is %.5f Ã‚Â± %0.2f" %
(CV_scores.mean(), CV_scores.std() * 2))


model = RandomForestClassifier(n_estimators= 80 ,max_depth=5 , max_features=8
,min_samples_split=3 ,random_state=7)
model.fit(X_train, y_train) predictions =
model.predict(X_test)
print_scores(model, X_train, y_train, predictions)
```

**Output:**

| Survived | Pclass | Sex | Age | SibSp | Parch | Fare | Embarked |
|---|---|---|---|---|---|---|---|
| 0 | 3 | 0 | 22 | 1 | 0 | 7.2500 | S |
| 1 | 1 | 1 | 38 | 1 | 0 | 71.2833 | C |
| 1 | 3 | 1 | 26 | 0 | 0 | 7.9250 | S |
| 1 | 1 | 1 | 35 | 1 | 0 | 53.1000 | S |
| 0 | 3 | 0 | 35 | 0 | 0 | 8.0500 | S |

**Conclusion:**

**Hence, we learn that, Machine Learning, Machine Learning Model, Training the Model, Testing the model, And Actual Prediction.**

.