1.Uber Ride

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn import metrics
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.cbook import boxplot_stats
import plotly.express as px
from geopy import Point, distance
from math import *
warnings.filterwarnings("ignore")

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

df.head()

   Unnamed: 0                            key  fare_amount  \
0    24238194    2015-05-07 19:52:06.0000003          7.5
1    27835199    2009-07-17 20:04:56.0000002          7.7
2    44984355   2009-08-24 21:45:00.00000061         12.9
3    25894730    2009-06-26 08:22:21.0000001          5.3
4    17610152  2014-08-28 17:47:00.000000188         16.0

           pickup_datetime  pickup_longitude  pickup_latitude  \
0  2015-05-07 19:52:06 UTC        -73.999817        40.738354
1  2009-07-17 20:04:56 UTC        -73.994355        40.728225
2  2009-08-24 21:45:00 UTC        -74.005043        40.740770
3  2009-06-26 08:22:21 UTC        -73.976124        40.790844
4  2014-08-28 17:47:00 UTC        -73.925023        40.744085

   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.999512         40.723217                1
1         -73.994710         40.750325                1
2         -73.962565         40.772647                1
3         -73.965316         40.803349                3
4         -73.973082         40.761247                5

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

df.head()

   fare_amount          pickup_datetime  pickup_longitude
pickup_latitude  \
0          7.5  2015-05-07 19:52:06 UTC        -73.999817
40.738354
1          7.7  2009-07-17 20:04:56 UTC        -73.994355
40.728225
```

```
2          12.9  2009-08-24 21:45:00 UTC        -74.005043
40.740770
3           5.3  2009-06-26 08:22:21 UTC        -73.976124
40.790844
4          16.0  2014-08-28 17:47:00 UTC        -73.925023
40.744085

   dropoff_longitude  dropoff_latitude  passenger_count
0         -73.999512         40.723217                1
1         -73.994710         40.750325                1
2         -73.962565         40.772647                1
3         -73.965316         40.803349                3
4         -73.973082         40.761247                5
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 193786 entries, 0 to 199999
Data columns (total 14 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        193786 non-null  float64
 1   pickup_datetime    193786 non-null  datetime64[ns, UTC]
 2   pickup_longitude   193786 non-null  float64
 3   pickup_latitude    193786 non-null  float64
 4   dropoff_longitude  193786 non-null  float64
 5   dropoff_latitude   193786 non-null  float64
 6   passenger_count    193786 non-null  int64
 7   distance_km        193786 non-null  float64
 8   pickup_hr          193786 non-null  int64
 9   day                193786 non-null  int64
 10  month              193786 non-null  int64
 11  year               193786 non-null  int64
 12  day_of_week        193786 non-null  int64
 13  day_name           193786 non-null  object
dtypes: datetime64[ns, UTC](1), float64(6), int64(6), object(1)
memory usage: 22.2+ MB
```

```
df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"],
errors="coerce")
```

```
df.describe().T
```

```
                      count       mean        std          min
25%  \
fare_amount        200000.0  11.359955   9.901776   -52.000000
6.000000
pickup_longitude   200000.0 -72.527638  11.437787 -1340.648410 -
73.992065
pickup_latitude    200000.0  39.935885   7.720539   -74.015515
```

```
40.734796
dropoff_longitude  199999.0 -72.525292  13.117408 -3356.666300 -
73.991407
dropoff_latitude   199999.0  39.923890   6.794829  -881.985513
40.733823
passenger_count    200000.0   1.684535   1.385997     0.000000
1.000000

                         50%        75%          max
fare_amount         8.500000  12.500000   499.000000
pickup_longitude  -73.981823 -73.967154    57.418457
pickup_latitude    40.752592  40.767158  1644.421482
dropoff_longitude -73.980093 -73.963658  1153.572603
dropoff_latitude   40.753042  40.768001   872.697628
passenger_count     1.000000   2.000000   208.000000
```

```
df.isna().sum()
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

```
df = df.dropna()
```

```
def distance_transform(longitude1, latitude1, longitude2, latitude2):
    distance = []
    for pos in range(len(longitude1)):
        long1,lati1,long2,lati2 = map(radians,
[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]])
        dist_long = long2 - long1
        dist_lati = lati2 - lati1
        a = sin(dist_lati/2)**2 + cos(lati1) * cos(lati2) *
sin(dist_long/2)**2
        c = 2 * asin(sqrt(a))*6371
        distance.append(c)
    return distance
```

```
df["distance_km"] =
distance_transform(df["pickup_longitude"].to_numpy(),
df["pickup_latitude"].to_numpy(),

df["dropoff_longitude"].to_numpy(), df["dropoff_latitude"].to_numpy())
```

```
df = df.assign(pickup_hr = df.pickup_datetime.dt.hour,
               day= df.pickup_datetime.dt.day,
               month = df.pickup_datetime.dt.month,
```

```
               year = df.pickup_datetime.dt.year,
               day_of_week = df.pickup_datetime.dt.dayofweek,
               day_name=df.pickup_datetime.dt.day_name())

df.head()

   fare_amount           pickup_datetime  pickup_longitude
pickup_latitude  \
0          7.5 2015-05-07 19:52:06+00:00        -73.999817
40.738354
1          7.7 2009-07-17 20:04:56+00:00        -73.994355
40.728225
2         12.9 2009-08-24 21:45:00+00:00        -74.005043
40.740770
3          5.3 2009-06-26 08:22:21+00:00        -73.976124
40.790844
4         16.0 2014-08-28 17:47:00+00:00        -73.925023
40.744085

   dropoff_longitude  dropoff_latitude  passenger_count
distance_km  \
0         -73.999512         40.723217                1     1.683323

1         -73.994710         40.750325                1     2.457590

2         -73.962565         40.772647                1     5.036377

3         -73.965316         40.803349                3     1.661683

4         -73.973082         40.761247                5     4.475450


   pickup_hr  day  month  year  day_of_week  day_name
0         19    7      5  2015            3  Thursday
1         20   17      7  2009            4    Friday
2         21   24      8  2009            0    Monday
3          8   26      6  2009            4    Friday
4         17   28      8  2014            3  Thursday

def find_outliers(df):
   q1 = df.quantile(0.25)
   q3 = df.quantile(0.75)
   IQR = q3-q1
   outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
   return outliers

outliers = find_outliers(df['fare_amount'])
print('number of outliers:' + str(len(outliers)))
print('max outlier value:' + str(outliers.max()))
print('min outlier value:' + str(outliers.min()))
outliers
```

```
number of outliers:17166
max outlier value:499.0
min outlier value:-52.0
```

```
6         24.50
30        25.70
34        39.50
39        29.00
48        56.80
          ...
199976    49.70
199977    43.50
199982    57.33
199985    24.00
199997    30.90
Name: fare_amount, Length: 17166, dtype: float64
```

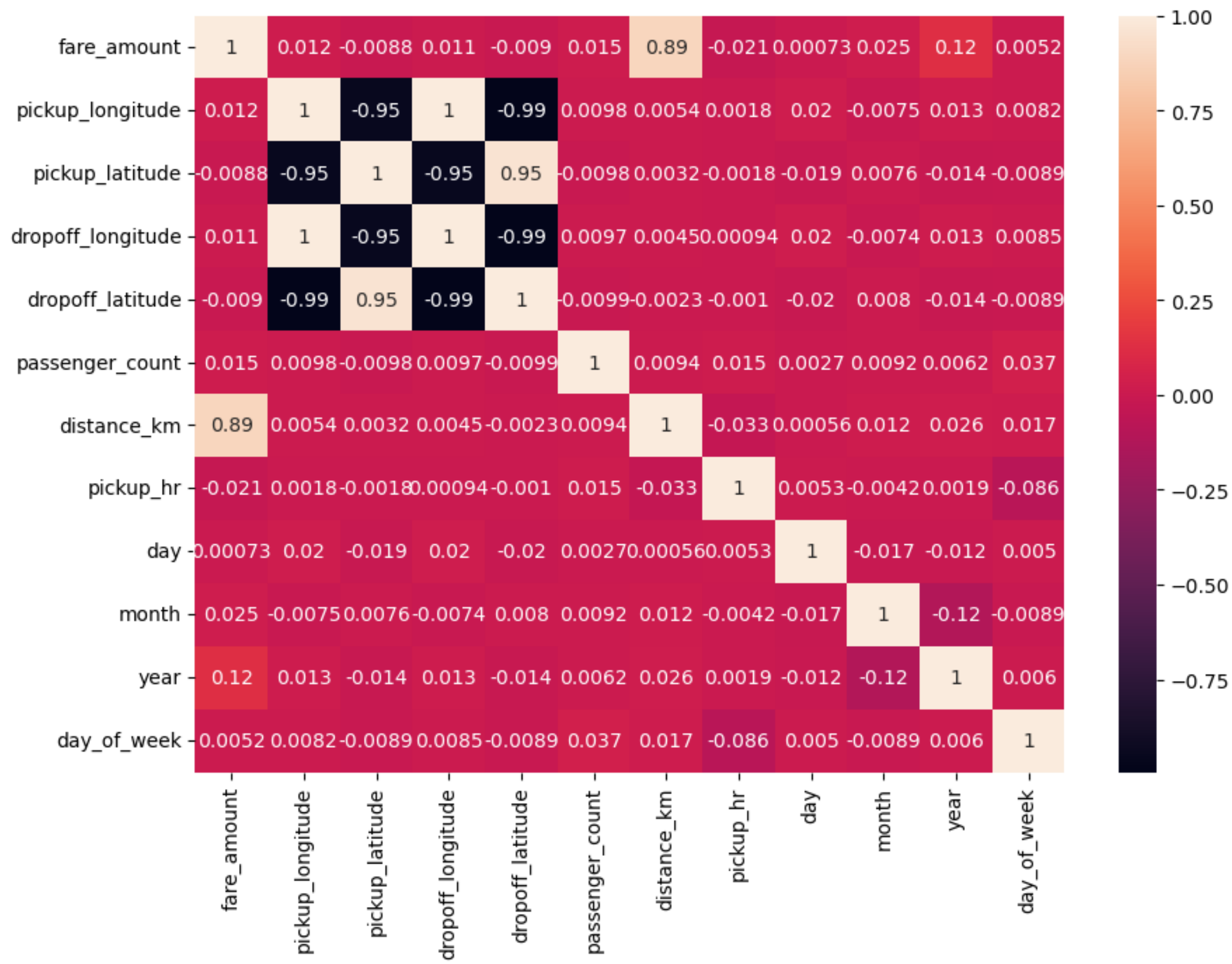
```
outliers = find_outliers(df['passenger_count'])
print('number of outliers:' + str(len(outliers)))
print('max outlier value:' + str(outliers.max()))
print('min outlier value:' + str(outliers.min()))
outliers
```

```
number of outliers:22557
max outlier value:208
min outlier value:4
```

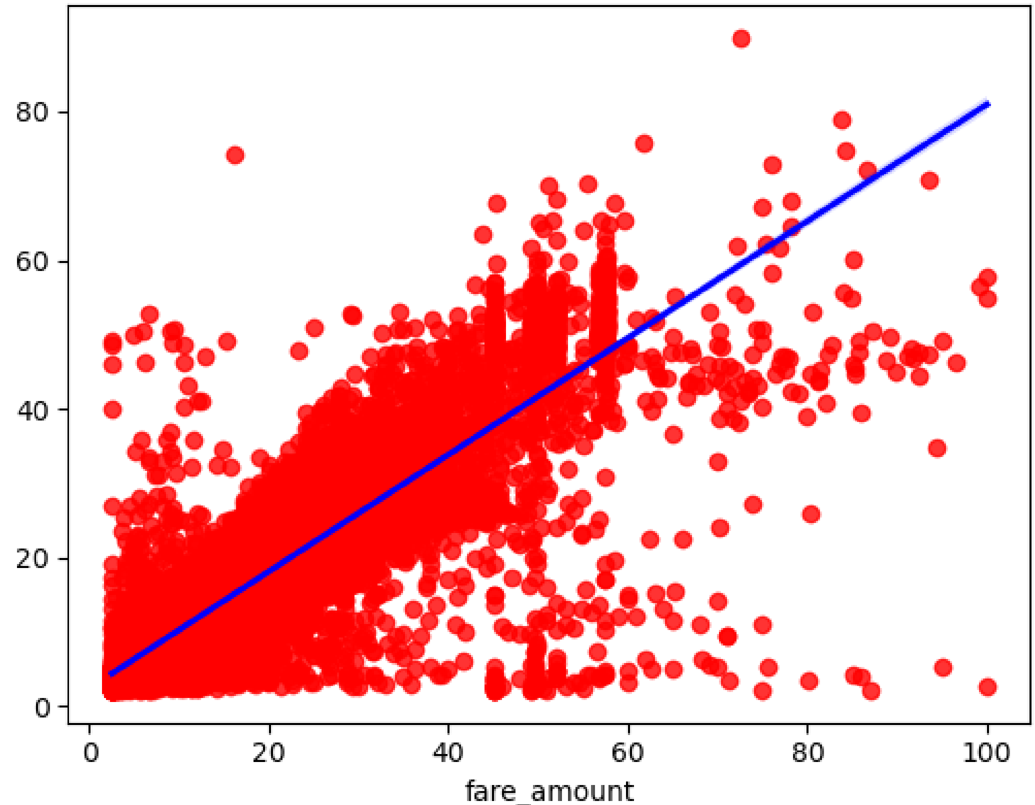
```
4         5
6         5
12        5
24        5
29        5
         ..
199958    5
199959    5
199962    4
199969    5
199985    5
Name: passenger_count, Length: 22557, dtype: int64
```

```
df.drop(df[df['distance_km'] == 0].index, inplace = True)
df.drop(df[df['distance_km'] > 60].index, inplace = True)
df.drop(df[df['fare_amount'] > 100].index, inplace = True)
df.drop(df[df['fare_amount'] < 0].index, inplace = True)
df.drop(df[df['passenger_count'] > 6].index, inplace = True)
```

```
plt.figure(figsize=(10,7))
sns.heatmap(df.corr(), annot=True)
plt.show()
```




```
x = df[["year", "distance_km"]]
y = df["fare_amount"]

scaler = StandardScaler()

x = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=42)

model = LinearRegression()
model.fit(x_train, y_train)

LinearRegression()

y_pred = model.predict(x_test)

sns.regplot(x=y_test, y=y_pred, color="red", line_kws={"color" :
"blue"})
plt.show()
```




```
print(f"Mean absolute error {metrics.mean_absolute_error(y_test,
y_pred)}")
print(f"Mean squared error {metrics.mean_squared_error(y_test,
y_pred)}")
print(f"Root mean squared error
{np.sqrt(metrics.mean_squared_error(y_test, y_pred))}")
```

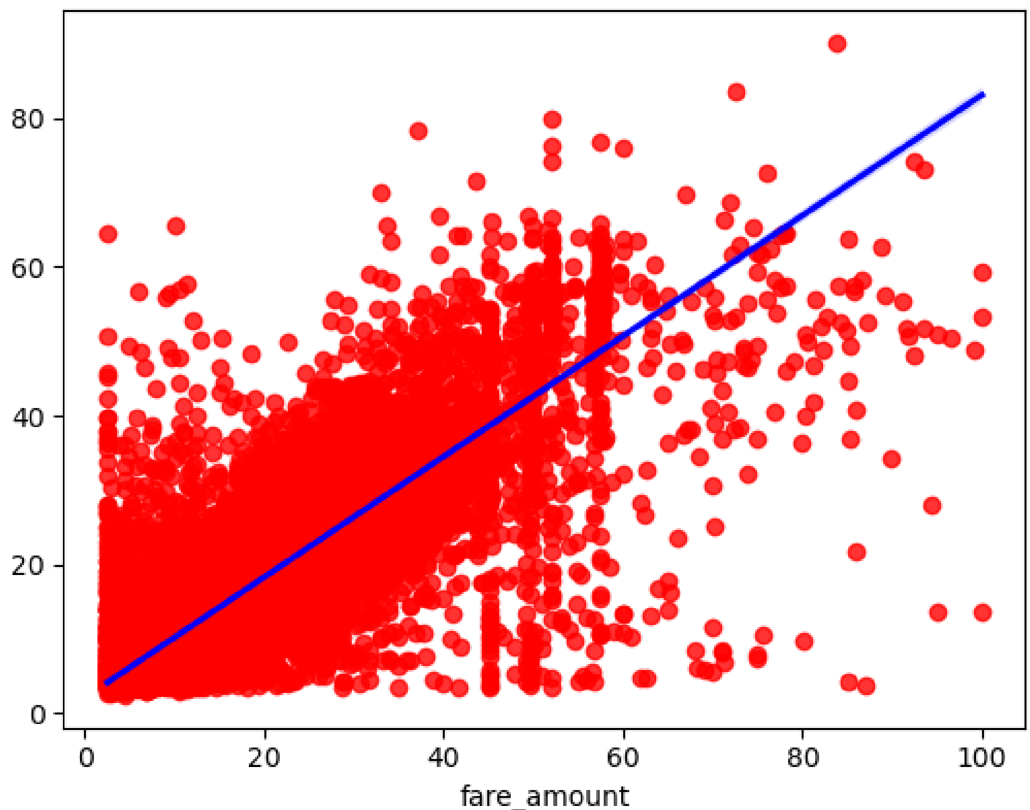
```
Mean absolute error 2.243879503459476
Mean squared error 18.32381910645457
Root mean squared error 4.280633026370582
```

```
model = RandomForestRegressor()
model.fit(x_train, y_train)
```

```
RandomForestRegressor()
```

```
y_pred = model.predict(x_test)
```

```
sns.regplot(x=y_test, y=y_pred, color="red", line_kws={"color" :
"blue"})
plt.show()
```




```
print(f"Mean absolute error {metrics.mean_absolute_error(y_test,
y_pred)}")
print(f"Mean squared error {metrics.mean_squared_error(y_test,
y_pred)}")
print(f"Root mean squared error
{np.sqrt(metrics.mean_squared_error(y_test, y_pred))}")
```

```
Mean absolute error 2.500337233122361
Mean squared error 21.283639254402114
Root mean squared error 4.613419475226821
```

```
def read_data(path: str) -> pd.DataFrame:
    """
    Read data from csv file.

    Args:
        path (str): path to csv file.

    Returns:
        pd.DataFrame: dataframe of csv file.
    """
    df = pd.read_csv(path)

    return df
```

2.Email Classification

```
def basic_info(df: pd.DataFrame) -> pd.DataFrame:
    """
    Get basic information of dataframe.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        pd.DataFrame: dataframe of basic information.
    """
    return df.info()

def distance_transform(longitude1: np.ndarray, latitude1: np.ndarray,
longitude2: np.ndarray, latitude2: np.ndarray) -> list:
    """
    Calculate distance between two points.

    Args:
        longitude1 (np.ndarray): array of longitude of first point.
        latitude1 (np.ndarray): array of latitude of first point.
        longitude2 (np.ndarray): array of longitude of second point.
        latitude2 (np.ndarray): array of latitude of second point.

    Returns:
        list: list of distance between two points.
    """
    distance = []
    for pos in range(len(longitude1)):
        long1,lati1,long2,lati2 = map(radians,
[longitude1[pos],latitude1[pos],longitude2[pos],latitude2[pos]])
        dist_long = long2 - long1
        dist_lati = lati2 - lati1
        a = sin(dist_lati/2)**2 + cos(lati1) * cos(lati2) *
sin(dist_long/2)**2
        c = 2 * asin(sqrt(a))*6371
        distance.append(c)

    return distance

def find_outliers(df: pd.DataFrame) -> pd.DataFrame:
    """
    Find outliers in dataframe.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        pd.DataFrame: dataframe of outliers.
    """
```

```
    q1 = df.quantile(0.25)
    q3 = df.quantile(0.75)
    IQR = q3-q1
    outliers = df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]

    return outliers

def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    """
    Preprocess dataframe.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        pd.DataFrame: dataframe after preprocessing.
    """
    df = df.drop(["Unnamed: 0", "key"], axis=1)
    df["pickup_datetime"] = pd.to_datetime(df["pickup_datetime"],
errors="coerce")
    df = df.dropna()
    df["distance_km"] =
distance_transform(df["pickup_longitude"].to_numpy(),
df["pickup_latitude"].to_numpy(),

df["dropoff_longitude"].to_numpy(), df["dropoff_latitude"].to_numpy())
    df = df.assign(pickup_hr = df.pickup_datetime.dt.hour,
               day= df.pickup_datetime.dt.day,
               month = df.pickup_datetime.dt.month,
               year = df.pickup_datetime.dt.year,
               day_of_week = df.pickup_datetime.dt.dayofweek,
               day_name=df.pickup_datetime.dt.day_name())
    outliers = find_outliers(df['fare_amount'])
    print('number of outliers for fare amount:' + str(len(outliers)))
    print('max outlier value for fare amount:' + str(outliers.max()))
    print('min outlier value for fare amount:' + str(outliers.min()))
    print(outliers)
    outliers = find_outliers(df['passenger_count'])
    print('number of outliers for fare amount:' + str(len(outliers)))
    print('max outlier value for fare amount:' + str(outliers.max()))
    print('min outlier value for fare amount:' + str(outliers.min()))
    print(outliers)
    df.drop(df[df['distance_km'] == 0].index, inplace = True)
    df.drop(df[df['distance_km'] > 60].index, inplace = True)
    df.drop(df[df['fare_amount'] > 100].index, inplace = True)
    df.drop(df[df['fare_amount'] < 0].index, inplace = True)
    df.drop(df[df['passenger_count'] > 6].index, inplace = True)

    return df
```

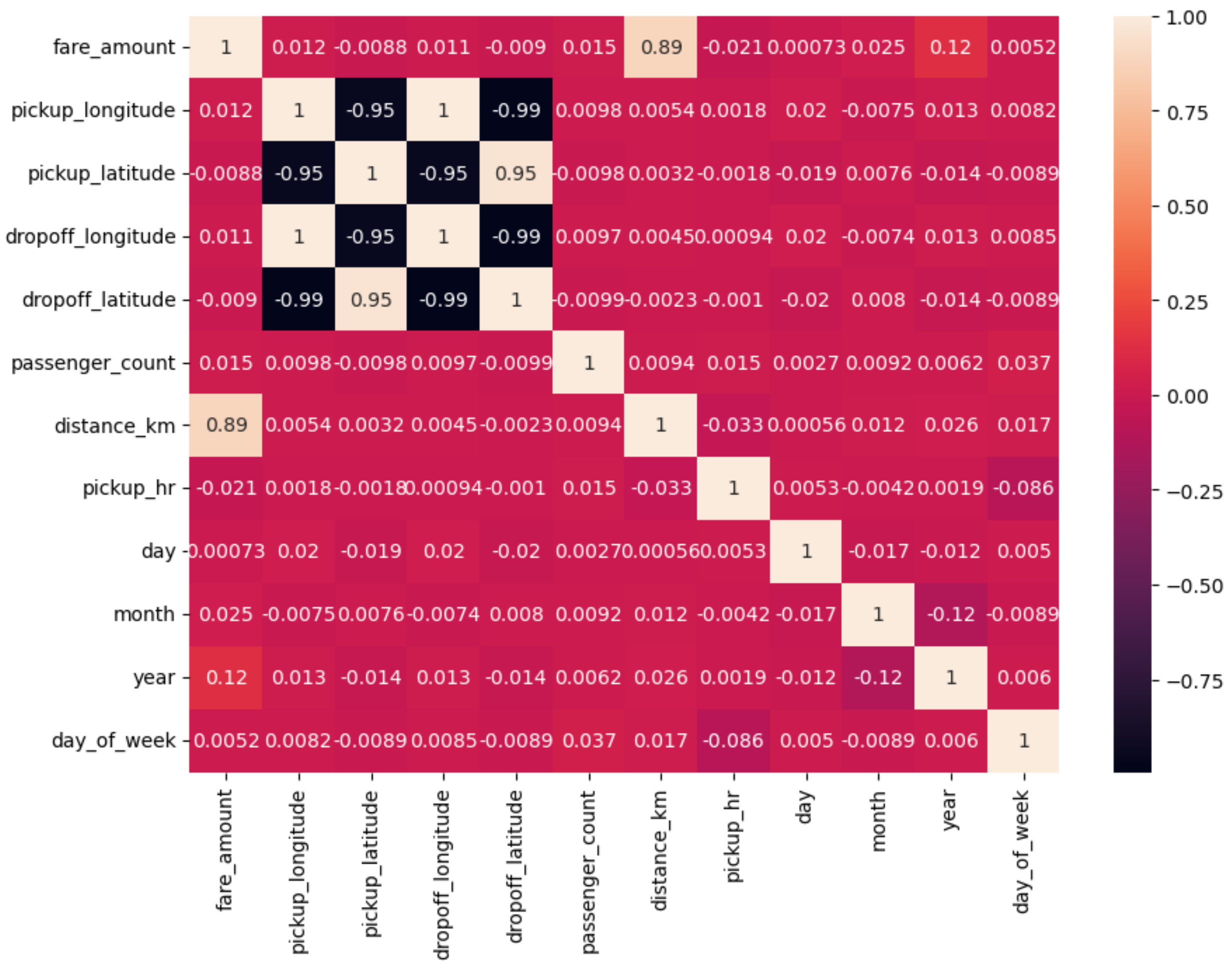
```python
def visualize_correlation(df: pd.DataFrame) -> None:
    """
    Visualize correlation between features.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        None.
    """
    plt.figure(figsize=(10,7))
    sns.heatmap(df.corr(), annot=True)
    plt.show()

def split_data(df: pd.DataFrame) -> tuple:
    """
    Split data into train and test set.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        tuple: tuple of train and test set.
    """
    x = df[["year", "distance_km"]]
    y = df["fare_amount"]
    scaler = StandardScaler()
    scaler.fit_transform(x)
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=42)

    return x_train, x_test, y_train, y_test

def create_model(model_name: str) -> object:
    """
    Create model.

    Args:
        model_name (str): name of model.

    Returns:
        object: model.
    """
    if model_name == "LR":
        model = LinearRegression()
    elif model_name == "RFR":
        model = RandomForestRegressor()

    return model
```

```
def train_model(model: object, x_train: np.ndarray, y_train:
np.ndarray) -> None:
    """
    Train model.

    Args:
        model (object): model.
        x_train (np.ndarray): array of train set.
        y_train (np.ndarray): array of train set.

    Returns:
        None.
    """
    model.fit(x_train, y_train)

def test_model(model: object, x_test: np.ndarray) -> np.ndarray:
    """
    Test model.

    Args:
        model (object): model.
        x_test (np.ndarray): array of test set.

    Returns:
        np.ndarray: array of predicted value.
    """
    y_pred = model.predict(x_test)

    return y_pred

def reg_line(y_test: np.ndarray, y_pred: np.ndarray) -> None:
    """
    Visualize regression line.

    Args:
        y_test (np.ndarray): test value.
        y_pred (np.ndarray): predicted value.
    """
    sns.regplot(x=y_test, y=y_pred, color="red", line_kws={"color" :
"blue"})
    plt.show()

def metrics_model(y_test: np.ndarray, y_pred: np.ndarray) -> None:
    """
    Calculate metrics of model.

    Args:
        y_test (np.ndarray): test value.
        y_pred (np.ndarray): predicted value.
```

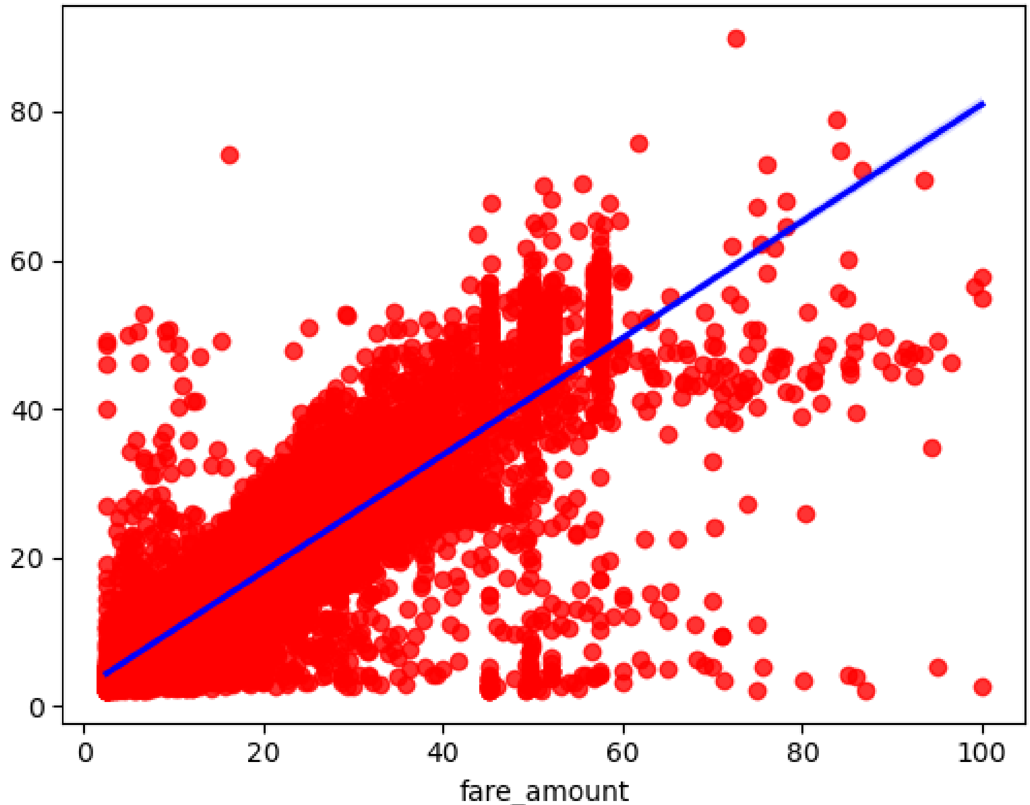
```
    Returns:
        None.
    """
    print(f"Mean absolute error {metrics.mean_absolute_error(y_test,
y_pred)}")
    print(f"Mean squared error {metrics.mean_squared_error(y_test,
y_pred)}")
    print(f"Root mean squared error
{np.sqrt(metrics.mean_squared_error(y_test, y_pred))}")

df = read_data("/kaggle/input/uber-fares-dataset/uber.csv")
print(basic_info(df))
df = preprocess(df)
print("\nCorrelation Matrix:\n")
visualize_correlation(df)
x_train, x_test, y_train, y_test = split_data(df)
model = create_model("LR")
train_model(model, x_train, y_train)
y_pred = test_model(model, x_test)
print("\nRegression Line:\n")
reg_line(y_test, y_pred)
print("\nModel Metrics:\n")
metrics_model(y_test, y_pred)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
None
number of outliers for fare amount:17166
max outlier value for fare amount:499.0
min outlier value for fare amount:-52.0
6         24.50
30        25.70
34        39.50
39        29.00
48        56.80
          ...
```

```
199976    49.70
199977    43.50
199982    57.33
199985    24.00
199997    30.90
Name: fare_amount, Length: 17166, dtype: float64
number of outliers for fare amount:22557
max outlier value for fare amount:208
min outlier value for fare amount:4
4         5
6         5
12        5
24        5
29        5
         ..
199958    5
199959    5
199962    4
199969    5
199985    5
Name: passenger_count, Length: 22557, dtype: int64

Correlation Matrix:
```



```
Regression Line:
```




```
Model Metrics:

Mean absolute error 2.2438795034594645
Mean squared error 18.323819106454575
Root mean squared error 4.280633026370583
```

3.Neutral Network Classifier

Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.utils import resample
from sklearn import metrics
from tqdm.notebook import tqdm
%matplotlib inline
warnings.filterwarnings("ignore")

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

df.head()

df.shape

df.describe().T
```

Without upsampling

```
df = df.drop("Email No.", axis=1)

df.isna().sum()

sns.distplot(x=df["Prediction"])
plt.show()

x = df.drop("Prediction", axis=1)
y = df[["Prediction"]]

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
```

KNN with elbow plot

```
k_values = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
accuracy_values = []

for i in tqdm(range(len(k_values))):
    model = KNeighborsClassifier(n_neighbors=k_values[i])
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)
```

```
accuracy_values

px.line(x=k_values, y=accuracy_values)

optimal_k = -1
optimal_accuracy = -1
for i in list(zip(k_values, accuracy_values)):
    if i[1] > optimal_accuracy:
        optimal_k = i[0]
        optimal_accuracy = i[1]

knn_model = KNeighborsClassifier(n_neighbors=optimal_k)

knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)

print(metrics.classification_report(y_test, y_pred))
```

SVM

```
svm_model = SVC()

svm_model.fit(x_train, y_train)

y_pred = svm_model.predict(x_test)

print(metrics.classification_report(y_test, y_pred))
```

With upsampling

```
spam_data = df[df["Prediction"] == 1]
ham_data = df[df["Prediction"] == 0]

spam_upsample = resample(spam_data,
             replace=True,
             n_samples=int(0.8*len(ham_data)),
             random_state=42)

new_df = ham_data
new_df = new_df.append(spam_upsample)

new_df.head()

new_df.shape

new_df = new_df.sample(frac=1)

sns.distplot(new_df["Prediction"])
plt.show()

x = new_df.drop("Prediction", axis=1)
y = new_df[["Prediction"]]
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.2)
```

KNN with elbow plot

```
k_values = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
accuracy_values = []

for i in tqdm(range(len(k_values))):
    model = KNeighborsClassifier(n_neighbors=k_values[i])
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)

px.line(x=k_values, y=accuracy_values)

optimal_k = -1
optimal_accuracy = -1
for i in list(zip(k_values, accuracy_values)):
    if i[1] > optimal_accuracy:
        optimal_k = i[0]
        optimal_accuracy = i[1]

knn_model = KNeighborsClassifier(n_neighbors=optimal_k)

knn_model.fit(x_train, y_train)

y_pred = knn_model.predict(x_test)

print(metrics.classification_report(y_test, y_pred))
```

SVM

```
svm_model = SVC()

svm_model.fit(x_train, y_train)

y_pred = svm_model.predict(x_test)

print(metrics.classification_report(y_test, y_pred))
```

Functions

```
def read_data(path: str) -> pd.DataFrame:
    """
    Read data from csv file.
```

```
    Args:
        path (str): path to csv file.

    Returns:
        pd.DataFrame: dataframe of csv file.
    """
    df = pd.read_csv(path)
    return df

def basic_info(df: pd.DataFrame) -> pd.DataFrame:
    """
    Get basic information of dataframe.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        pd.DataFrame: dataframe of basic information.
    """
    return df.info()

def preprocess(df: pd.DataFrame) -> pd.DataFrame:
    df = df.drop("Email No.", axis=1)
    return df

def split_data(df: pd.DataFrame) -> tuple:
    """
    Split data into train and test set.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        tuple: tuple of train and test set.
    """
    x = df.drop("Prediction", axis=1)
    y = df[["Prediction"]]
    x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3, random_state=42)
    return x_train, x_test, y_train, y_test

def knn_model_with_elbow_method(x_train: np.ndarray, x_test:
np.ndarray, y_train: np.ndarray, y_test: np.ndarray, k_values: list) -
> np.ndarray:
    """
    KNN model with elbow method.

    Args:
        x_train (np.ndarray): x_train data.
        x_test (np.ndarray): x_test data.
```

```
        y_train (np.ndarray): y_train data.
        y_test (np.ndarray): y_test data.
        k_values (list): list of k values.

    Returns:
        np.ndarray: y_pred data.
    """
    accuracy_values = []
    for i in tqdm(range(len(k_values))):
        model = KNeighborsClassifier(n_neighbors=k_values[i])
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)
        accuracy = metrics.accuracy_score(y_test, y_pred)
        accuracy_values.append(accuracy)
    fig = px.line(x=k_values, y=accuracy_values, title="K value vs
Accuracy")
    fig.update_layout(xaxis_title="K values", yaxis_title="Accuracy
values")
    fig.show()
    optimal_k = -1
    optimal_accuracy = -1
    for i in list(zip(k_values, accuracy_values)):
        if i[1] > optimal_accuracy:
            optimal_k = i[0]
            optimal_accuracy = i[1]
    knn_model = KNeighborsClassifier(n_neighbors=optimal_k)
    knn_model.fit(x_train, y_train)
    y_pred = knn_model.predict(x_test)
    return y_pred

def svm_model(x_train: np.ndarray, x_test: np.ndarray, y_train:
np.ndarray, y_test: np.ndarray) -> np.ndarray:
    """
    SVM model.

    Args:
        x_train (np.ndarray): x_train data.
        x_test (np.ndarray): x_test data.
        y_train (np.ndarray): y_train data.
        y_test (np.ndarray): y_test data.

    Returns:
        np.ndarray: y_pred data.
    """
    svm_model = SVC()
    svm_model.fit(x_train, y_train)
    y_pred = svm_model.predict(x_test)
    return y_pred

def metrics_report(y_test: np.ndarray, y_pred: np.ndarray) -> None:
```

```
    """
    Print metrics report.

    Args:
        y_test (np.ndarray): y_test data.
        y_pred (np.ndarray): y_pred data.
    """
    print(metrics.classification_report(y_test, y_pred))

def upsample_data(df: pd.DataFrame) -> pd.DataFrame:
    """
    Upsample data.

    Args:
        df (pd.DataFrame): dataframe.

    Returns:
        pd.DataFrame: upsampled dataframe.
    """
    spam_data = df[df["Prediction"] == 1]
    ham_data = df[df["Prediction"] == 0]
    spam_upsample = resample(
        spam_data,
        replace=True,
        n_samples=int(0.8*len(ham_data)),
        random_state=42
    )
    new_df = ham_data
    new_df = new_df.append(spam_upsample)
    new_df = new_df.sample(frac=1)
    return new_df

df =
read_data("/kaggle/input/email-spam-classification-dataset-csv/emails.
csv")
basic_info(df)
df = preprocess(df)
df = upsample_data(df)
x_train, x_test, y_train, y_test = split_data(df)
k_values = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29]
y_pred_knn = knn_model_with_elbow_method(x_train, x_test, y_train,
y_test, k_values)
y_pred_svm = svm_model(x_train, x_test, y_train, y_test)
print("Metrics for KNN-\n")
metrics_report(y_test, y_pred_knn)
print("Metrics for SVM-\n")
metrics_report(y_test, y_pred_svm)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5172 entries, 0 to 5171
```

```
Columns: 3002 entries, Email No. to Prediction
dtypes: int64(3001), object(1)
memory usage: 118.5+ MB
```

{"model_id":"47e76ce383a84e9bb256a446d794f3f0","version_major":2,"version_minor":0}

```
Metrics for KNN-

              precision    recall  f1-score   support

           0       0.96      0.87      0.91      1128
           1       0.85      0.96      0.90       855

    accuracy                           0.91      1983
   macro avg       0.90      0.91      0.91      1983
weighted avg       0.91      0.91      0.91      1983

Metrics for SVM-

              precision    recall  f1-score   support

           0       0.79      0.92      0.85      1128
           1       0.86      0.67      0.76       855

    accuracy                           0.81      1983
   macro avg       0.82      0.80      0.80      1983
weighted avg       0.82      0.81      0.81      1983
```

4.Gradient Descent Algorithm

Implement Gradient Descent Algorithm to find the local minima of a function. For example, find the local minima of the function y=(x+3)**2 starting from the point x=2.

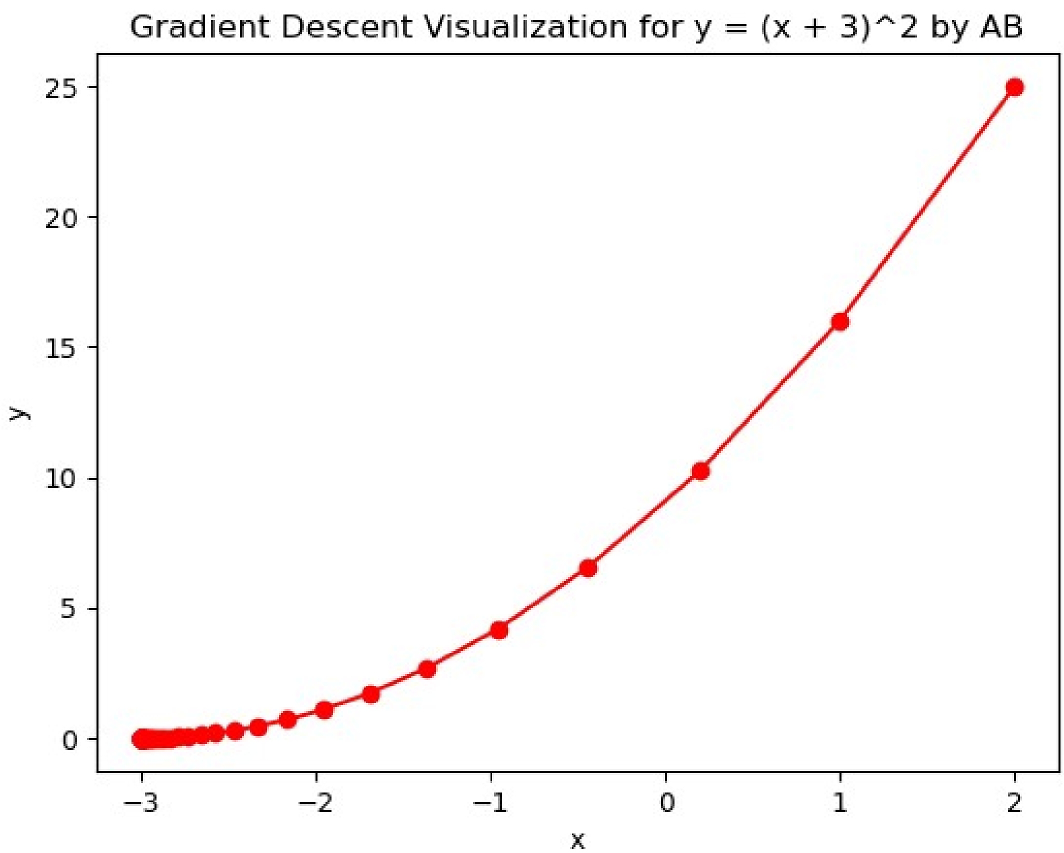
```
import matplotlib.pyplot as plt

def cost_function(x):
    # ithe given function yenar
    return (x + 3) ** 2

def gradient(x):
    # ithe derivate of given function yenar
    return 2 * (x + 3)

learning_rate = 0.1
initial_x = 2.0
num_iterations = 100

x_values = []
y_values = []
x = initial_x
for i in range(num_iterations):
    x_values.append(x)
    y_values.append(cost_function(x))
    gradient_value = gradient(x)
    x = x - learning_rate * gradient_value

    print(f'Iteration {i+1}: x = {x}, Cost = {cost_function(x)}')

print(f'Optimal x: {x}')
```

```
Iteration 1: x = 1.0, Cost = 16.0
Iteration 2: x = 0.19999999999999996, Cost = 10.240000000000002
Iteration 3: x = -0.44000000000000017, Cost = 6.553599999999998
Iteration 4: x = -0.9520000000000001, Cost = 4.194304
Iteration 5: x = -1.3616000000000001, Cost = 2.6843545599999996
Iteration 6: x = -1.6892800000000001, Cost = 1.7179869183999996
Iteration 7: x = -1.951424, Cost = 1.099511627776
Iteration 8: x = -2.1611392, Cost = 0.7036874417766399
Iteration 9: x = -2.32891136, Cost = 0.4503599627370493
Iteration 10: x = -2.463129088, Cost = 0.28823037615171165
Iteration 11: x = -2.5705032704, Cost = 0.1844674407370954
Iteration 12: x = -2.6564026163200003, Cost = 0.11805916207174093
Iteration 13: x = -2.725122093056, Cost = 0.07555786372591429
Iteration 14: x = -2.7800976744448, Cost = 0.04835703278458515
Iteration 15: x = -2.82407813955584, Cost = 0.030948500982134555
Iteration 16: x = -2.8592625116446717, Cost = 0.019807040628566166
Iteration 17: x = -2.8874100093157375, Cost = 0.012676506002282305
Iteration 18: x = -2.90992800745259, Cost = 0.008112963841460692
Iteration 19: x = -2.927942405962072, Cost = 0.005192296858534868
Iteration 20: x = -2.9423539247696575, Cost = 0.0033230699894623056
```

```
Iteration 21: x = -2.953883139815726, Cost = 0.002126764793255884
Iteration 22: x = -2.9631065118525806, Cost = 0.0013611294676837786
Iteration 23: x = -2.9704852094820646, Cost = 0.0008711228593176078
Iteration 24: x = -2.9763881675856516, Cost = 0.0005575186299632732
Iteration 25: x = -2.981110534068521, Cost = 0.00035681192317650156
Iteration 26: x = -2.984888427254817, Cost = 0.00022835963083295564
Iteration 27: x = -2.9879107418038537, Cost = 0.00014615016373308945
Iteration 28: x = -2.990328593443083, Cost = 9.353610478917726e-05
Iteration 29: x = -2.9922628747544664, Cost = 5.986310706507345e-05
Iteration 30: x = -2.993810299803573, Cost = 3.83123885216492e-05
Iteration 31: x = -2.995048239842858, Cost = 2.451992865385725e-05
Iteration 32: x = -2.9960385918742864, Cost = 1.5692754338469342e-05
Iteration 33: x = -2.9968308734994293, Cost = 1.0043362776619253e-05
Iteration 34: x = -2.9974646987995435, Cost = 6.427752177036323e-06
Iteration 35: x = -2.997971759039635, Cost = 4.113761393302886e-06
Iteration 36: x = -2.998377407231708, Cost = 2.6328072917135587e-06
Iteration 37: x = -2.998701925785366, Cost = 1.6849966666971388e-06
Iteration 38: x = -2.998961540628293, Cost = 1.0783978666865378e-06
Iteration 39: x = -2.9991692325026342, Cost = 6.901746346793842e-07
Iteration 40: x = -2.9993353860021075, Cost = 4.417117661946878e-07
Iteration 41: x = -2.999468308801686, Cost = 2.826955303647891e-07
Iteration 42: x = -2.9995746470413485, Cost = 1.8092513943361614e-07
Iteration 43: x = -2.9996597176330786, Cost = 1.1579208923763523e-07
Iteration 44: x = -2.999727774106463, Cost = 7.410693711203819e-08
Iteration 45: x = -2.99978221928517, Cost = 4.7428439751781807e-08
Iteration 46: x = -2.9998257754281363, Cost = 3.035420144107846e-08
Iteration 47: x = -2.999860620342509, Cost = 1.9426688922339734e-08
Iteration 48: x = -2.999888496274007, Cost = 1.243308091029743e-08
Iteration 49: x = -2.9999107970192056, Cost = 7.9571717826062e-09
Iteration 50: x = -2.9999286376153647, Cost = 5.092589940842615e-09
Iteration 51: x = -2.9999429100922916, Cost = 3.259257562149415e-09
Iteration 52: x = -2.999954328073833, Cost = 2.0859248397837384e-09
Iteration 53: x = -2.9999634624590668, Cost = 1.3349918974486118e-09
Iteration 54: x = -2.9999707699672533, Cost = 8.543948143723039e-10
Iteration 55: x = -2.999976615973803, Cost = 5.468126811899669e-10
Iteration 56: x = -2.9999812927790424, Cost = 3.499601159582557e-10
Iteration 57: x = -2.9999850342232337, Cost = 2.2397447421860056e-10
Iteration 58: x = -2.999988027378587, Cost = 1.433436634977776e-10
Iteration 59: x = -2.9999904219028695, Cost = 9.173994464198049e-11
Iteration 60: x = -2.9999923375222957, Cost = 5.871356456950638e-11
Iteration 61: x = -2.9999938700178364, Cost = 3.757668132666189e-11
Iteration 62: x = -2.999995096014269, Cost = 2.4049076048192486e-11
Iteration 63: x = -2.9999960768114153, Cost = 1.5391408670843192e-11
Iteration 64: x = -2.9999968614491324, Cost = 9.850501548782124e-12
Iteration 65: x = -2.999997489159306, Cost = 6.3043209907745444e-12
Iteration 66: x = -2.9999979913274446, Cost = 4.034765434809332e-12
Iteration 67: x = -2.9999983930619556, Cost = 2.5822498785634223e-12
Iteration 68: x = -2.9999987144495646, Cost = 1.6526399220522305e-12
Iteration 69: x = -2.9999989715596516, Cost = 1.0576895502961154e-12
```

```
Iteration 70: x = -2.9999991772477212, Cost = 6.769213121895138e-13
Iteration 71: x = -2.999999341798177, Cost = 4.3322963956744853e-13
Iteration 72: x = -2.9999994734385416, Cost = 2.7726696951023927e-13
Iteration 73: x = -2.9999995787508333, Cost = 1.7745086041172427e-13
Iteration 74: x = -2.9999996630006667, Cost = 1.1356855066350352e-13
Iteration 75: x = -2.9999997304005332, Cost = 7.268387247253274e-14
Iteration 76: x = -2.9999997843204267, Cost = 4.651767834410857e-14
Iteration 77: x = -2.9999998274563415, Cost = 2.977131407892966e-14
Iteration 78: x = -2.9999998619650734, Cost = 1.9053640961475125e-14
Iteration 79: x = -2.9999998895720585, Cost = 1.2194330254575965e-14
Iteration 80: x = -2.999999911657647, Cost = 7.804371331543109e-15
Iteration 81: x = -2.9999999293261177, Cost = 4.994797639633387e-15
Iteration 82: x = -2.999999943460894, Cost = 3.1966704893653676e-15
Iteration 83: x = -2.9999999547687155, Cost = 2.045869097124455e-15
Iteration 84: x = -2.9999999638149726, Cost = 1.309356209304147e-15
Iteration 85: x = -2.9999999710519782, Cost = 8.379879636702507e-16
Iteration 86: x = -2.9999999768415826, Cost = 5.363122967489605e-16
Iteration 87: x = -2.999999981473266, Cost = 3.432398699193347e-16
Iteration 88: x = -2.9999999851786128, Cost = 2.1967351938118145e-16
Iteration 89: x = -2.99999998814289, Cost = 1.4059105661644777e-16
Iteration 90: x = -2.999999990514312, Cost = 8.997827286453327e-17
Iteration 91: x = -2.9999999924114498, Cost = 5.758609463330129e-17
Iteration 92: x = -2.9999999939291597, Cost = 3.685510164371068e-17
Iteration 93: x = -2.9999999951433276, Cost = 2.358726677741145e-17
Iteration 94: x = -2.999999996114662, Cost = 1.5095850047368678e-17
Iteration 95: x = -2.99999999689173, Cost = 9.661342926036557e-18
Iteration 96: x = -2.9999999975133838, Cost = 6.18326035608692e-18
Iteration 97: x = -2.999999998010707, Cost = 3.957287334634505e-18
Iteration 98: x = -2.9999999984085655, Cost = 2.532663894166083e-18
Iteration 99: x = -2.999999998726852, Cost = 1.6209053445792253e-18
Iteration 100: x = -2.999999998981482, Cost = 1.0373792396055266e-18
Optimal x: -2.999999998981482
```

```python
plt.plot(x_values, y_values, 'ro-')
plt.title('Gradient Descent Visualization for y = (x + 3)^2 by AB')
plt.xlabel('x')
plt.ylabel('y')
plt.show()
```


Gradient Descent Visualization for y = (x + 3)^2 by AB
y
x
25
20
15
10
5
0
−3
−2
−1
0
1
2

5.K-Nearest NEighbors

Import libraries

```
import pandas as pd
import numpy as np
import plotly.express as px
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.utils import resample
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from mlxtend.plotting import plot_confusion_matrix
from tqdm.notebook import tqdm
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Data loading and preprocessing

```
df = pd.read_csv("diabetes.csv")

df

     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI
\
0              6      148             72             35        0  33.6

1              1       85             66             29        0  26.6

2              8      183             64              0        0  23.3

3              1       89             66             23       94  28.1

4              0      137             40             35      168  43.1

..           ...      ...            ...            ...      ...   ...

763           10      101             76             48      180  32.9

764            2      122             70             27        0  36.8

765            5      121             72             23      112  26.2

766            1      126             60              0        0  30.1

767            1       93             70             31        0  30.4
```

```
     Pedigree  Age  Outcome
0       0.627   50        1
1       0.351   31        0
2       0.672   32        1
3       0.167   21        0
4       2.288   33        1
..        ...  ...      ...
763     0.171   63        0
764     0.340   27        0
765     0.245   30        0
766     0.349   47        1
767     0.315   23        0

[768 rows x 9 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Pregnancies    768 non-null    int64
 1   Glucose        768 non-null    int64
 2   BloodPressure  768 non-null    int64
 3   SkinThickness  768 non-null    int64
 4   Insulin        768 non-null    int64
 5   BMI            768 non-null    float64
 6   Pedigree       768 non-null    float64
 7   Age            768 non-null    int64
 8   Outcome        768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
df.describe().T
```

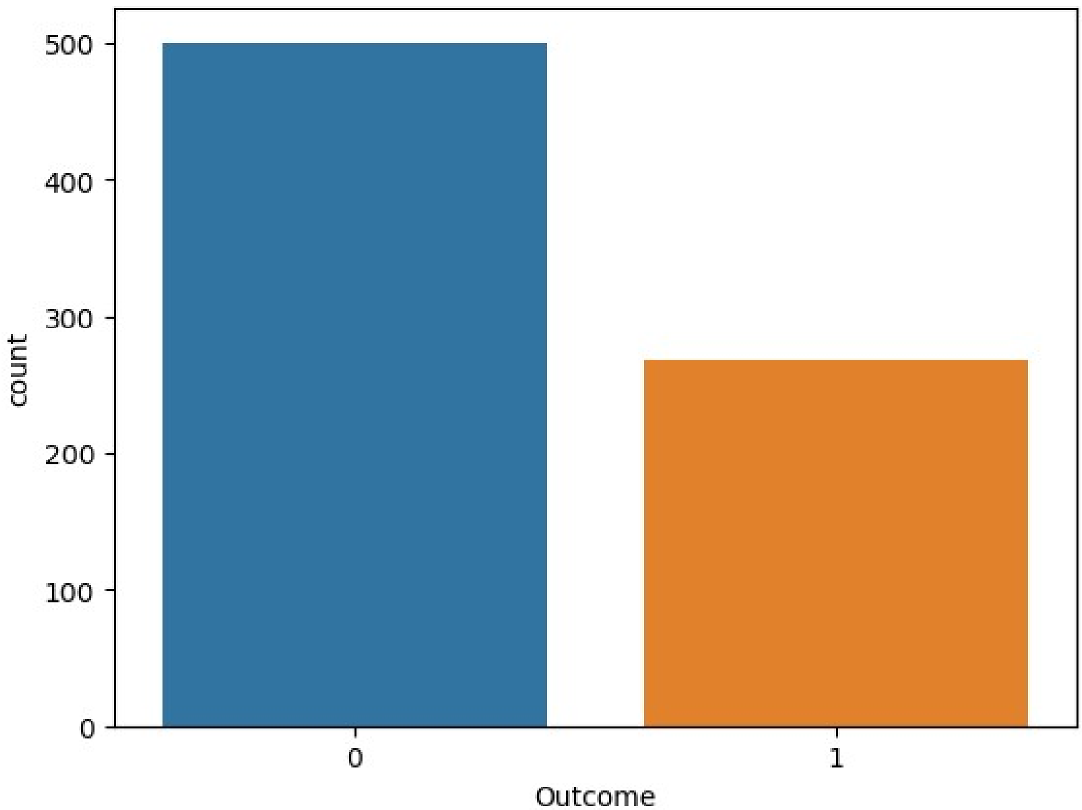
```
               count        mean         std    min       25%
50%  \
Pregnancies    768.0    3.845052    3.369578  0.000   1.00000
3.0000
Glucose        768.0  120.894531   31.972618  0.000  99.00000
117.0000
BloodPressure  768.0   69.105469   19.355807  0.000  62.00000
72.0000
SkinThickness  768.0   20.536458   15.952218  0.000   0.00000
23.0000
Insulin        768.0   79.799479  115.244002  0.000   0.00000
30.5000
BMI            768.0   31.992578    7.884160  0.000  27.30000
```

```
32.0000
Pedigree        768.0    0.471876    0.331329   0.078   0.24375
0.3725
Age             768.0   33.240885   11.760232  21.000  24.00000
29.0000
Outcome         768.0    0.348958    0.476951   0.000   0.00000
0.0000

                      75%     max
Pregnancies       6.00000   17.00
Glucose         140.25000  199.00
BloodPressure    80.00000  122.00
SkinThickness    32.00000   99.00
Insulin         127.25000  846.00
BMI              36.60000   67.10
Pedigree          0.62625    2.42
Age              41.00000   81.00
Outcome           1.00000    1.00
```

```
df["Outcome"].value_counts()
```

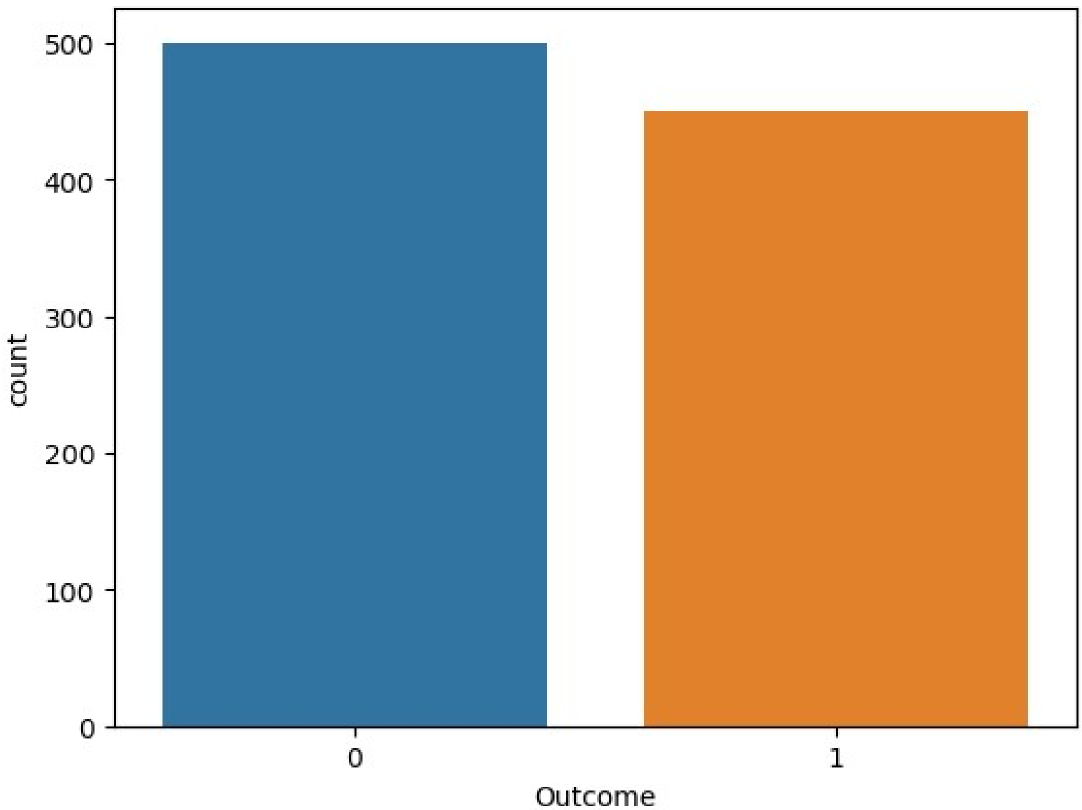
```
0    500
1    268
Name: Outcome, dtype: int64
```

```
sns.countplot(data=df, x=df["Outcome"])
plt.show()
```



Upsampling

```
negative_data = df[df["Outcome"] == 0]
positive_data = df[df["Outcome"] == 1]

positive_upsample = resample(positive_data,
                             replace=True,
                             n_samples=int(0.9*len(negative_data)),
                             random_state=42)

new_df = negative_data
new_df = new_df.append(positive_upsample)

new_df.shape

(950, 9)

new_df = new_df.sample(frac=1)

sns.countplot(data=new_df, x=new_df["Outcome"])
plt.show()
```




```
x = new_df.drop("Outcome", axis=1)
y = new_df[["Outcome"]]

scaler = MinMaxScaler()
scaled_values = scaler.fit_transform(x)

x_train, x_test, y_train, y_test = train_test_split(scaled_values, y,
test_size=0.2)
```

KNN with elbow plot

```
k_values = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31,
33, 35, 37, 39, 41, 43, 45, 47, 49]
accuracy_values = []

for i in tqdm(range(len(k_values))):
    model = KNeighborsClassifier(n_neighbors=k_values[i])
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    accuracy = metrics.accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)
```

{"model_id":"bacfa33a6b4e4ce8833eaf0ffdb45be7","version_major":2,"version_minor":0}

```
px.line(x=k_values, y=accuracy_values)
```

```
optimal_k = -1
optimal_accuracy = -1
for i in list(zip(k_values, accuracy_values)):
    if i[1] > optimal_accuracy:
        optimal_k = i[0]
        optimal_accuracy = i[1]
```

```
knn_model = KNeighborsClassifier(n_neighbors=optimal_k)
```

```
knn_model.fit(x_train, y_train)
```

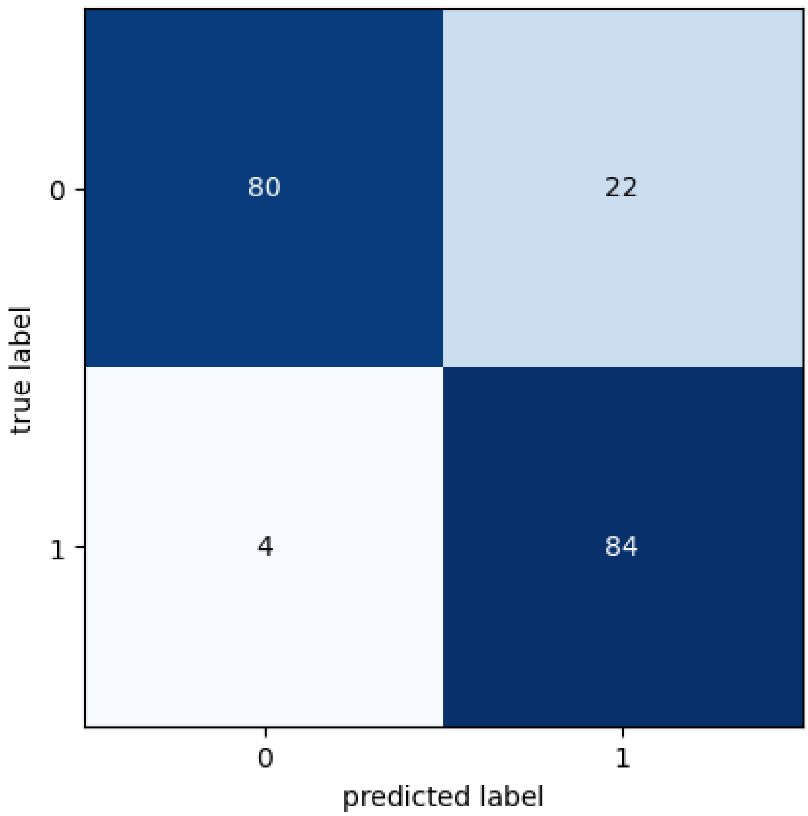
```
KNeighborsClassifier(n_neighbors=1)
```

```
y_pred = knn_model.predict(x_test)
```

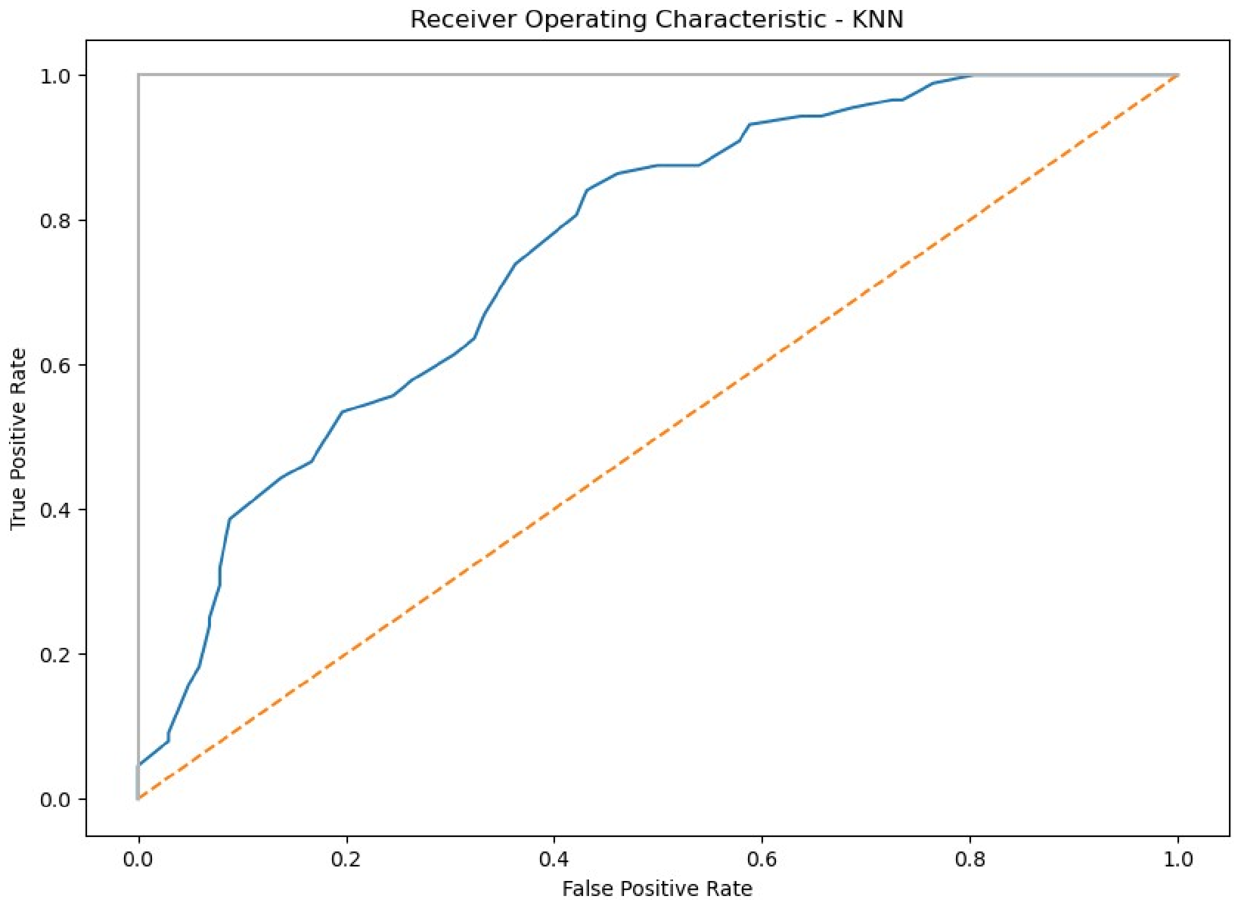
```
print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.95      0.78      0.86       102
           1       0.79      0.95      0.87        88

    accuracy                           0.86       190
   macro avg       0.87      0.87      0.86       190
weighted avg       0.88      0.86      0.86       190
```

```
cm = metrics.confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cm)
plt.show()
```



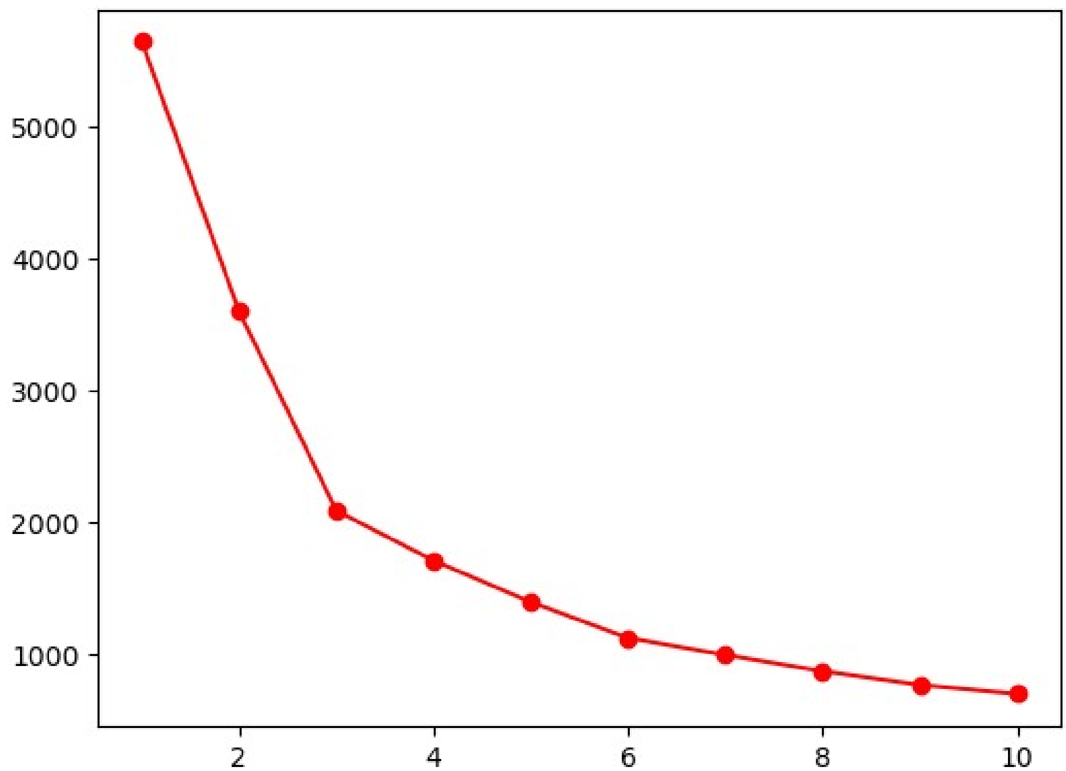

```
y_score = model.predict_proba(x_test)[:,1]

false_positive_rate, true_positive_rate, threshold =
metrics.roc_curve(y_test, y_score)

print('roc_auc_score for DecisionTree: ',
metrics.roc_auc_score(y_test, y_score))

roc_auc_score for DecisionTree:  0.7575200534759358

plt.subplots(1, figsize=(10,7))
plt.title('Receiver Operating Characteristic - KNN')
plt.plot(false_positive_rate, true_positive_rate)
plt.plot([0, 1], ls="--")
plt.plot([0, 0], [1, 0] , c=".7"), plt.plot([1, 1] , c=".7")
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```


Receiver Operating Characteristic - KNN
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

```python
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
import warnings
from sklearn.preprocessing import StandardScaler
warnings.filterwarnings('ignore')

df = pd.read_csv("sales_data_sample.csv", encoding="latin")

df.head()
```

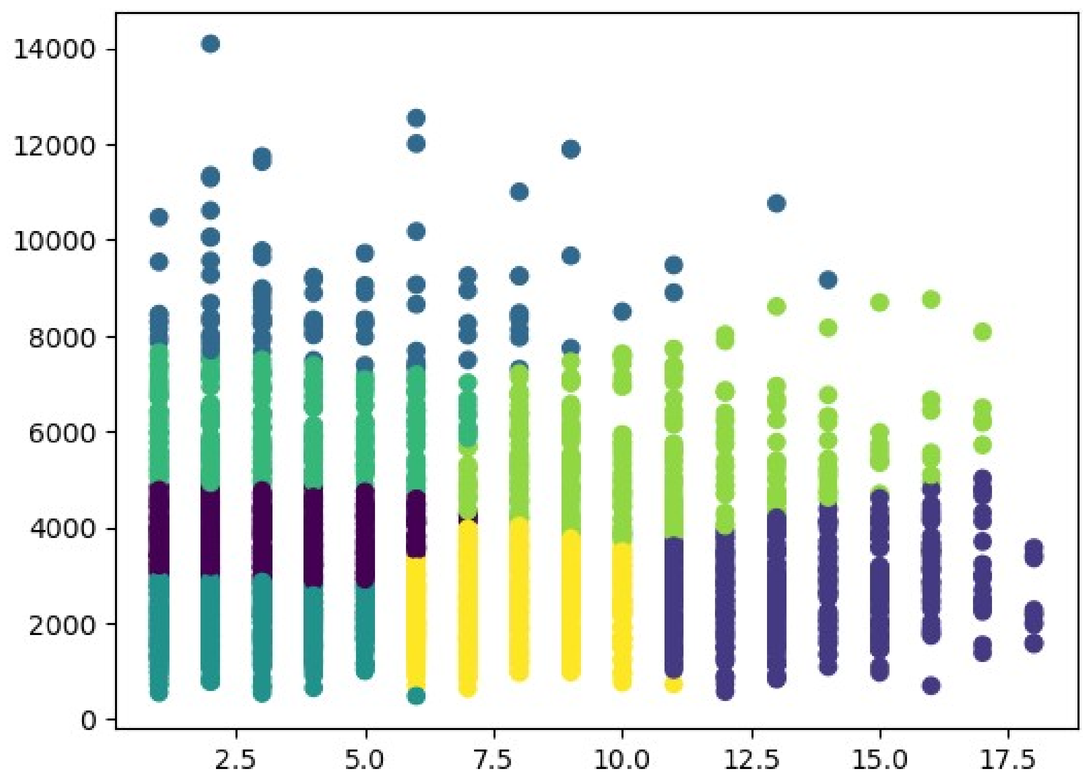
```
   ORDERNUMBER  QUANTITYORDERED  PRICEEACH  ORDERLINENUMBER
SALES  \
0        10107               30      95.70                2  2871.00

1        10121               34      81.35                5  2765.90

2        10134               41      94.74                2  3884.34

3        10145               45      83.26                6  3746.70

4        10159               49     100.00               14  5205.27


         ORDERDATE   STATUS  QTR_ID  MONTH_ID  YEAR_ID  ...  \
0   2/24/2003 0:00  Shipped       1         2     2003  ...
1    5/7/2003 0:00  Shipped       2         5     2003  ...
2    7/1/2003 0:00  Shipped       3         7     2003  ...
3   8/25/2003 0:00  Shipped       3         8     2003  ...
4  10/10/2003 0:00  Shipped       4        10     2003  ...

                   ADDRESSLINE1  ADDRESSLINE2           CITY STATE  \
0        897 Long Airport Avenue           NaN            NYC    NY
1             59 rue de l'Abbaye           NaN          Reims   NaN
2  27 rue du Colonel Pierre Avia           NaN          Paris   NaN
3             78934 Hillside Dr.           NaN       Pasadena    CA
4                7734 Strong St.           NaN  San Francisco    CA

  POSTALCODE COUNTRY TERRITORY CONTACTLASTNAME CONTACTFIRSTNAME
DEALSIZE
0      10022     USA       NaN              Yu             Kwai
Small
1      51100  France      EMEA         Henriot             Paul
Small
2      75508  France      EMEA        Da Cunha           Daniel
Medium
3      90003     USA       NaN           Young            Julie
Medium
4        NaN     USA       NaN           Brown            Julie
Medium
```

```
[5 rows x 25 columns]

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2823 entries, 0 to 2822
Data columns (total 25 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ORDERNUMBER       2823 non-null   int64  
 1   QUANTITYORDERED   2823 non-null   int64  
 2   PRICEEACH         2823 non-null   float64
 3   ORDERLINENUMBER   2823 non-null   int64  
 4   SALES             2823 non-null   float64
 5   ORDERDATE         2823 non-null   object 
 6   STATUS            2823 non-null   object 
 7   QTR_ID            2823 non-null   int64  
 8   MONTH_ID          2823 non-null   int64  
 9   YEAR_ID           2823 non-null   int64  
 10  PRODUCTLINE       2823 non-null   object 
 11  MSRP              2823 non-null   int64  
 12  PRODUCTCODE       2823 non-null   object 
 13  CUSTOMERNAME      2823 non-null   object 
 14  PHONE             2823 non-null   object 
 15  ADDRESSLINE1      2823 non-null   object 
 16  ADDRESSLINE2      302 non-null    object 
 17  CITY              2823 non-null   object 
 18  STATE             1337 non-null   object 
 19  POSTALCODE        2747 non-null   object 
 20  COUNTRY           2823 non-null   object 
 21  TERRITORY         1749 non-null   object 
 22  CONTACTLASTNAME   2823 non-null   object 
 23  CONTACTFIRSTNAME  2823 non-null   object 
 24  DEALSIZE          2823 non-null   object 
dtypes: float64(2), int64(7), object(16)
memory usage: 551.5+ KB

df = df[['ORDERLINENUMBER', 'SALES']]

scaler = StandardScaler()
scaled_values = scaler.fit_transform(df.values)

wcss = []
for i in range(1, 11):
    model = KMeans(n_clusters=i, init='k-means++')
    model.fit_predict(scaled_values)
    wcss.append(model.inertia_)

plt.plot(range(1, 11), wcss, 'ro-')
plt.show()
```




```
model = KMeans(n_clusters=7, init='k-means++')
clusters = model.fit_predict(scaled_values)
clusters

array([3, 3, 0, ..., 4, 3, 6])

df['cluster'] = clusters

df

      ORDERLINENUMBER    SALES  cluster
0                   2  2871.00        3
1                   5  2765.90        3
2                   2  3884.34        0
3                   6  3746.70        0
4                  14  5205.27        5
...               ...      ...      ...
2818               15  2244.40        1
2819                1  3978.51        0
2820                4  5417.57        4
2821                1  2116.16        3
2822                9  3079.44        6

[2823 rows x 3 columns]

model.inertia_
```

```
993.4283577026391

plt.scatter(df['ORDERLINENUMBER'], df['SALES'], c=df['cluster'])
plt.show()
```



SCTR's Pune Institute of Computer Technology
Dhankawadi, Pune

A MINI-PROJECT REPORT ON

Titanic Survivor Prediction Using Machine Learning

SUBMITTED BY

Aayush Goyal	41302
Divya khandare	41321
Arnav Firke	41322

Under the guidance of
Prof. V.S. Gaikwad

DEPARTMENT OF COMPUTER ENGINEERING
ACADEMIC YEAR 2024-25

Contents

1 Title

Titanic Survivor Prediction Using Machine Learning

2 Introduction

Predicting Titanic survivors is a significant application of machine learning in predictive analytics. Understanding the factors influencing passenger survival can offer valuable insights into historical events and enhance decision-making processes in safety and emergency response systems. This project focuses on building a machine learning model to predict the likelihood of a Titanic passenger's survival based on various features such as age, gender, class, and fare.

The motivation behind this project arises from the historical significance of the Titanic disaster and the wealth of data available to explore survival patterns. By leveraging advanced machine learning algorithms and Titanic passenger data, we aim to create a predictive model that can accurately determine survival outcomes based on passenger characteristics.

We selected algorithms like Logistic Regression, Random Forest, and Support Vector Machines (SVM) for their ability to capture complex relationships between features and their proven effectiveness in classification tasks. By training these models, we aim to uncover the key factors that contributed to survival and provide interpretable results.

The significance of this project lies in its contribution to understanding historical data through modern machine-learning techniques. This approach not only aids in predictive accuracy but also sheds light on critical variables that impacted survival during one of history's most well-known maritime disasters.

3 Problem Statement

To predict Titanic Survivors based on various given features.

4 About Dataset

The dataset used for Titanic survivor prediction is the well-known Titanic.csv which contains detailed information about the passengers aboard the RMS Titanic. This dataset originates from the records of the Titanic disaster and includes various features that describe the passengers, such as their age, gender, ticket class, fare, and whether or not they survived.

The dataset provides a comprehensive snapshot of the 2,224 passengers on board, with data on both survivors and non-survivors, allowing for in-depth analysis. The data covers essential variables like Pclass (ticket class), Sex, Age, SibSp(number of siblings/spouses aboard), Parch (number of parents/children aboard), Fare, Embarked (port of embarkation), and Survived (the target variable indicating survival).

This historical data allows us to explore the relationships between different features and survival outcomes, offering the potential to build a robust predictive model.

5 Objectives and Scope

5.1 Objectives of Project:

The main objective of the Titanic survivor prediction project is to accurately predict whether a passenger survived or not based on given input variables. By analyzing features such as age, gender, ticket class, and fare, this prediction can provide insights into the factors influencing survival during the Titanic disaster. The model can be used in historical analysis, educational purposes, and to better understand the relationship between passenger attributes and survival outcomes.

5.2 Scope of Project:

1. **Data Collection:** Utilize the Titanic dataset, which contains historical information about passengers aboard the Titanic, including variables such as age, gender, class, fare, and survival status. This data will be used for training and testing the machine learning model.

2. **Feature Selection:** Identify and select key features that have a significant impact on survival. This may include variables like passenger class (Pclass), gender (Sex), age, number of siblings/spouses aboard (SibSp), and fare, among others.

3. **Data Preprocessing:** Prepare the dataset by handling missing values, converting categorical data into numerical form (such as gender and embarked location), and scaling or normalizing features if needed to ensure proper model performance.

4. **Model Training:** Train the predictive model using machine learning algorithms such as Logistic Regression, Random Forest, and Support Vector Machines (SVM). These algorithms will analyze the selected features to predict the survival of each passenger.

5. **Model Evaluation:** Evaluate the performance of the trained model using appropriate metrics like accuracy, precision, recall, F1-score, and confusion matrix. Cross-validation and testing on unseen data will ensure the model's reliability.

6. **Model Deployment:** Deploy the trained model into an application that can take passenger data as input and predict the likelihood of survival. This system can be integrated into educational tools, simulations, or analytical software for further exploration of the Titanic disaster.

6 Methodological Details

This is a methodological breakdown of the project:

1. **Data Collection and Loading:** The Titanic survival prediction project begins with the collection of historical data, specifically the Titanic dataset, which is loaded using **Pandas**. The dataset includes essential features such as the age, gender, and class of the passengers, as well as whether they survived or not. The data was sourced from a `.csv` file stored in a cloud directory and was accessed and manipulated within a Python-based Jupyter notebook.

2. **Data Cleaning and Preprocessing:** Once the dataset is loaded, an initial inspection is carried out to identify missing values. The dataset includes several columns that are either irrelevant or contain excessive missing data, such as `PassengerId`, `Name`, `Ticket`, and `Cabin`, which are subsequently dropped. The remaining columns are retained for analysis.

3. **Exploratory Data Analysis:** EDA is conducted using visualizations created with **Seaborn** and **Matplotlib** to understand the distribution of survival across different features like gender (`Sex`) and passenger class (`Pclass`). The missing values in the `Age` and `Embarked` columns are handled through imputation. The `Age` column is imputed using the mean strategy, while the `Embarked` column is filled with its most frequent value (`mode`).

4. **Encoding Categorical Variables:** To facilitate machine learning model training, the categorical columns such as `Sex` and `Embarked` are converted into numerical formats using **Label Encoding**. This conversion is crucial for allowing algorithms to interpret categorical data during the training process.

5. **Feature Selection and Correlation Analysis:** Before proceeding to model development, a heatmap is generated to visualize the correlation matrix of the selected features. This step helps identify the relationships between different variables and survival, aiding in feature selection.

6. **Model Training:** The dataset is then split into training and testing sets using **train_test_split** from **Scikit-learn**, with 80% of the data used for training and 20% for testing. This allows the model to learn from a portion of the data while preserving another portion for evaluation purposes. The primary machine learning model used in this project is a **Random Forest Classifier** due to its robustness and ability to handle non-linear relationships. The model is trained on the selected features using the training set.

7. **Model Evaluation:** After training, predictions are made on the test set using the trained model. The model's performance is evaluated using metrics such as **accuracy**, **confusion matrix**, and **classification report**, which provide a detailed breakdown of how well the model is performing in terms of precision, recall, and F1-score.

7 Modern Engineering Tools Used

The breakdown of the tools and technologies used in the project:

7.1 Google Colab:

Google Colab is used as the primary platform for developing and running the project. It provides a Jupyter notebook environment with access to free GPU and TPU resources, which is useful for machine learning tasks.

7.2 Python:

Python is the programming language used for coding the project. It is a versatile language with rich libraries for data analysis, machine learning, and visualization.

7.3 Libraries/Frameworks:

- **NumPy:** Used for numerical computations and array manipulation.
- **Pandas:** Used for data manipulation and analysis, including reading and writing data in various formats.
- **Matplotlib:** Used for data visualization, including plotting graphs and charts.
- **scikit-learn:** Used for machine learning tasks such as model building, evaluation, and preprocessing.

Hardware Requirements:

Since Google Colab is a cloud-based platform, no specific hardware requirements are needed from the user's end. However, having a stable internet connection is necessary to access and work on Google Colab.

These libraries provide extensive documentation and resources for data analysis, visualization, and machine learning tasks.

Google Colab Environment:

Google Colab provides a pre-configured environment with the following specifications:

- Python runtime environment
- Access to GPU and TPU resources for accelerated computation
- Integrated code editor with support for Jupyter notebooks
- Ability to install additional libraries using pip or conda
- Integration with Google Drive for data storage and access

8 Conclusion

In conclusion, this project has successfully demonstrated the application of machine learning techniques for predicting the survival of Titanic passengers based on various features such as age, gender, ticket class, and fare. Through comprehensive data preprocessing, feature selection, model training, and evaluation, we have developed a predictive model capable of accurately forecasting passenger survival with solid performance metrics.

Machine learning algorithms like Logistic Regression, Random Forest, and Support Vector Machines (SVM) have proven to be effective in capturing the complex relationships between passenger characteristics and survival outcomes. By leveraging historical Titanic data and advanced machine learning methodologies, we gained valuable insights into the factors influencing survival during the disaster.

The practical applications of this predictive model extend to educational tools, historical analysis, and safety simulations, providing a better understanding of survival factors in maritime disasters. The model's predictions can serve as a foundation for further exploration of survival patterns and emergency preparedness.

Moving forward, opportunities for refinement include incorporating additional features such as cabin location or family size, improving model performance through hyperparameter tuning, and deploying the model into real-world simulations for broader evaluation.

Overall, this project represents a significant contribution toward analyzing historical data using machine learning, and it showcases the potential of data-driven approaches to understanding human survival in critical scenarios like the Titanic disaster.