ML_File

1.	Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks: 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. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset

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

data = pd.read_csv("/content/uber.csv")

data.head()

data.tail()

data.info()

data.describe()

data.shape

data.columns

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

data.head()

data['months'] = data['pickup_datetime']
data['hours'] = data['pickup_datetime']
data['months'] = data['months'].str.slice(start=5,stop=7)
```

```
data['hours'] = data['hours'].str.slice(start=11, stop=13)

data.head()

data = data.drop('pickup_datetime', axis=1)
data

# Alternate to datetime separation
# df = pd.read_csv("/content/uber.csv")
# df['pickup_datetime'] = pd.to_datetime(df['pickup_datetime'])
# df['hour'] = df['pickup_datetime'].dt.hour
# df['day'] = df['pickup_datetime'].dt.dayofweek
# df['month'] = df['pickup_datetime'].dt.month
# df = df.drop(columns=['pickup_datetime'])
# df

data.isnull().sum()

data['dropoff_latitude'] =
data['dropoff_latitude'].fillna(data['dropoff_latitude'].mean())
data['dropoff_longitude'] =
data['dropoff_longitude'].fillna(data['dropoff_longitude'].mean())
data.isnull().sum()

data.describe()

data.replace(to_replace=0, value = data['passenger_count'].mean(),
inplace=True)
data[data['fare_amount']<=0] = data['fare_amount'].mean()
data.describe()

data.plot(kind="box", subplots=True, layout=(6,2), figsize=(15,20))

def remove_outlier(df1, col):
  Q1 = df1[col].quantile(0.25)
  Q3 = df1[col].quantile(0.75)
  IQR = Q3-Q1
  lb = Q1 - 1.5*IQR
  ub = Q3 + 1.5*IQR
  data[col] = np.clip(df1[col], lb, ub)
```

```
  return df1

def treat_outliers_all(df1, colslist):
  for col in colslist:
    remove_outlier(df1, col)
  return df1

data.columns

cols = ['fare_amount', 'pickup_longitude', 'pickup_latitude',
'dropoff_longitude', 'dropoff_latitude', 'passenger_count']
treat_outliers_all(data, cols)

# --- Outliers using your exact syntax ---
# Q1 = y.quantile(0.25)
# Q3 = y.quantile(0.75)
# IQR = Q3 - Q1

# lower_bound = Q1 - 1.5 * IQR
# upper_bound = Q3 + 1.5 * IQR
# outliers = df[(df['fare_amount'] < lower_bound) | (df['fare_amount'] >
upper_bound)]
# print(f"Number of outliers: {len(outliers)}")

data.plot(kind="box", subplots=True, layout=(6,2), figsize=(15,20))

data.shape

data.isnull().sum()

plt.scatter(data['pickup_latitude'],data['fare_amount'])
plt.xlabel("pickup_latitude")
plt.ylabel("fare_amount")

plt.scatter(data['pickup_longitude'],data['fare_amount'])
plt.xlabel("pickup_longitude")
plt.ylabel("fare_amount")

plt.scatter(data['dropoff_latitude'],data['fare_amount'])
plt.xlabel("dropoff_latitude")
```

```
plt.ylabel("fare_amount")

plt.scatter(data['dropoff_longitude'],data['fare_amount'])
plt.xlabel("dropoff_longitude")
plt.ylabel("fare_amount")

corr_matrix = data.corr()
corr_matrix

sns.heatmap(corr_matrix, annot=True)

data.columns

X = data.iloc[:,1:]
y = data.iloc[:,0]

# OR
# features = ['pickup_longitude', 'pickup_latitude',
#             'dropoff_longitude', 'dropoff_latitude', 'passenger_count',
'hour', 'month', 'day']
# X = df[features]
# y = df['fare_amount']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,
random_state=42)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

lr = LinearRegression()
lr.fit(X_train, y_train)
lr.score(X_test,y_test)

y_pred = lr.predict(X_test)
result = pd.DataFrame()
result['Actual'], result['Predicted'] = y_test, y_pred
result.sample(10)

print('Mean Absolute Error:', metrics.mean_absolute_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)))

rf = RandomForestRegressor(n_estimators=10, random_state=42)
rf.fit(X_train, y_train)
rf.score(X_test,y_test)

y_pred = rf.predict(X_test)
result1 = pd.DataFrame()
result1['Actual'], result1['Predicted'] = y_test, y_pred
result1.sample(10)

print('Mean Absolute Error:', metrics.mean_absolute_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)))
```

2.	Classify the email using the binary classification method. Email Spam detection has two states: a) Normal State – Not Spam, b) Abnormal State – Spam. Use K-Nearest Neighbors and Support Vector Machine for classification. Analyze their performance. Dataset link: The emails.csv dataset on the Kaggle https://www.kaggle.com/datasets/balaka18/email-spam-classification-dataset-csv

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, ConfusionMatrixDisplay, precision_score, recall_score
import time
import matplotlib.pyplot as plt

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

data

data.head()

data.shape

data = data.drop('Email No.',axis=1)

data.shape
```

```
data.describe()

data.info()

data['Prediction'].value_counts()

X = data.drop('Prediction', axis=1)
y = data['Prediction']

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =
0.20, random_state = 42)

from sklearn.neighbors import KNeighborsClassifier
start = time.time()
neigh = KNeighborsClassifier(n_neighbors=2)
neigh.fit(X_train, y_train)

y_pred = neigh.predict(X_test)
knn_time = time.time() - start
knn_acc= accuracy_score(y_test, y_pred)

print("\n************** K-Nearest Neighbors ****************\n")
print(f"Training time: {knn_time}s")
print(f"Accuracy: {knn_acc}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

neigh.score(X_train, y_train)
neigh.score(X_test, y_test)

print("Confusion Matrix: ")
cm = confusion_matrix(y_test, y_pred)
cm

mat = ConfusionMatrixDisplay(confusion_matrix = cm)
mat.plot()
plt.show()
```

```
print("accuracy_score: ")
accuracy_score(y_test, y_pred)

print("precision_score: ")
precision_score(y_test, y_pred)

print("recall_score: ")
recall_score(y_test, y_pred)

print("Error: ")
1-accuracy_score(y_test, y_pred)

from sklearn.svm import SVC
start = time.time()
svm = SVC(kernel = 'linear', random_state=42)
svm.fit(X_train, y_train)

y_pred = svm.predict(X_test)
svm_time = time.time()-start
svm_acc= accuracy_score(y_test, y_pred)

print(f"Training time: {svm_time}s")
print(f"Accuracy: {svm_acc}")
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

svm.score(X_train, y_train)
svm.score(X_test, y_test)

print("Confusion Matrix: ")
cm = confusion_matrix(y_test, y_pred)
cm

mat = ConfusionMatrixDisplay(confusion_matrix = cm)
mat.plot()
plt.show()
```

4.	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 $x=2$.

```python
import numpy as np
import matplotlib.pyplot as plt

# Take user input for coefficients of quadratic ax^2 + bx + c
a = float(input("Enter coefficient a: "))
b = float(input("Enter coefficient b: "))
c = float(input("Enter coefficient c: "))
x0 = float(input("Enter starting x: "))
lr = float(input("Enter learning rate: "))
iters = int(input("Enter number of iterations: "))

def f(x):
  return a*x**2 + b*x + c

def grad(x):
  return 2*a*x + b

x=x0
path = [x]
for _ in range(iters):
  x = x - lr*grad(x)
  path.append(x)

x_plot = np.linspace(x0-10,x0+10,100)
y_plot = f(x_plot)
plt.plot(x_plot, y_plot, label='f(x)')
plt.plot(path, f(np.array(path)),'ro-',label='GD Path')
plt.title('Gradient Descent Algorithm')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.grid(True)
plt.show()

print('Local minimum at: x=',path[-1],' y=',f(path[-1]))

# import matplotlib.pyplot as plt
# import numpy as np
# def func1(x):
#   return (x+3)**2
```

```
# def gradient_func1(x):
#    return 2*(x+3)

# def gradient_descent(function, start, learn_rate, n_iter = 100,
tolerance = 0.1):
#     gradient = gradient_func1
#     function = func1
#     points = [start]
#     iters = 0

#     while iters < n_iter:
#         prev_x = start
#         start = start - learn_rate * gradient(prev_x)
#         iters = iters+1
#         points.append(start)
#     print("The local minimum occurs at", start)

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

#     fig = plt.figure(figsize = (10, 10))
#     plt.plot(x_, y, 'g')
#     plt.plot(points, function(np.array(points)), '-o')

#     plt.show()

# gradient_descent(function = func1, start = 2.0, learn_rate = 0.2, n_iter
= 50)
```

6.	Implement K-Means clustering/ hierarchical clustering on sales_data_sample.csv dataset. Determine the number of clusters using the elbow method. Dataset link : https://www.kaggle.com/datasets/kyanyoga/sample-sales-data

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.cluster import KMeans
```

```python
df = pd.read_csv('/content/sales_data_sample.csv',encoding='latin')
df.head()

df.describe()

df.info()

df['STATUS'] = LabelEncoder().fit_transform(df['STATUS'])
df['DEALSIZE'] = LabelEncoder().fit_transform(df['DEALSIZE'])
df.info()

df.columns

features = [
    'QUANTITYORDERED', 'PRICEEACH', 'SALES', 'MSRP',
    'ORDERLINENUMBER', 'QTR_ID', 'MONTH_ID', 'YEAR_ID',
    'STATUS', 'DEALSIZE'
]
X = df[features]
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Elbow method to find optimal number of clusters
inertias = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(X_scaled)
    inertias.append(kmeans.inertia_)

# Plot the elbow curve
plt.plot(range(1, 11), inertias, marker='o')
plt.title('Elbow Method for Optimal Clusters')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.grid(True)
plt.show()

# Fit KMeans with chosen number of clusters (e.g., 3)
kmeans = KMeans(n_clusters=3, random_state=42)
df['Cluster'] = kmeans.fit_predict(X_scaled)
```

```
# View sample results
print(df[['QUANTITYORDERED', 'PRICEEACH', 'SALES', 'MSRP',
'Cluster']].head())

plt.figure(figsize=(8,6))
plt.scatter(df['PRICEEACH'], df['SALES'], c=df['Cluster'], cmap='viridis',
s=50)
plt.title('Customer Segments based on SALES and PRICEEACH')
plt.xlabel('SALES')
plt.ylabel('PRICEEACH')
plt.colorbar(label='Cluster')
plt.show()
```