1)

```
import numpy as np # numpy does numbers
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.style.use('dark_background')
def f(x, noise):
	return(x**2 + 1 + noise)
x_train = np.linspace(-2, 2, 10)
noise_train = np.random.normal(0, 0.5, size = 10)
y_train = f(x_train, noise_train)
plt.scatter(x_train, y_train)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Training Data')
plt.show()
```

```
model1 = np.poly1d(np.polyfit(x_train, y_train, deg = 2))
model2 = np.poly1d(np.polyfit(x_train, y_train, deg = 9))
model3 = np.poly1d(np.polyfit(x_train, y_train, deg = 1))
plot_x = np.linspace(-2, 2, 100) # so we can plot a curve
plt.scatter(x_train, y_train, label='Actual Points')
plt.plot(plot_x, model1(plot_x), label = 'Polynomial model of degree 2', color =
'red')
plt.plot(plot_x, model2(plot_x), label = 'Polynomial model of degree 9', color =
'orange')
plt.plot(plot_x, model3(plot_x), label = 'Polynomial model of degree 1', color =
'green')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Training Data')
plt.legend()
fig = plt.gcf()
fig.set_dpi(200)
plt.show()
```

```
x_test = np.linspace(-2, 2, 100)
noise_test = np.random.normal(0, 0.5, size = 100)

y1cap = model1(x_test)
y2cap = model2(x_test)
y3cap = model3(x_test)
y_test = f(x_test, noise_test) # final exam answer key

y1_error = np.sum(np.abs(y_test - y1cap))
y2_error = np.sum(np.abs(y_test - y2cap))
y3_error = np.sum(np.abs(y_test - y3cap))
print("Error on Test dataset----")
print("Polynomial model of degree 2:", y1_error)
print("Polynomial model of degree 9:", y2_error)
print("Polynomial model of degree 1:", y3_error)
plt.scatter(x_test, y_test, label = 'Actual Points')
plt.plot(x_test, y1cap, label = 'Polynomial model of degree 2', color = 'red')
plt.plot(x_test, y2cap, label = 'Polynomial model of degree 9', color = 'orange')
plt.plot(x_test, y3cap, label = 'Polynomial model of degree 1', color = 'green')
plt.xlabel('x')
plt.ylabel('y')
plt.title('Performance on Test dataset')
plt.legend()
plt.show()
```

2)

```
import pandas as pd
df = pd.read_csv('auto_mpg.data', header = None, sep = '\s+')
df.columns = ['mpg', 'cylinders', 'displacement', 'horsepower', 'weight',
'acceleration', 'model year', 'origin', 'car name’]
df.head()

X = df.drop(['mpg','car name', 'model year', 'origin', 'horsepower'], axis = 1)
y = df['mpg'].values
print(X.shape)
print(y.shape)
X.head()
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
```

Multiple Linear Regression

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
lr = LinearRegression()
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)

print("R-Squared: ", r2_score(y_test, y_pred))
print("Weights: ", lr.coef_)
print("Constant: ", lr.intercept_)
```

Ridge Regression

```
from sklearn.linear_model import
Ridge
rr = Ridge(alpha = 100)
rr.fit(X_train, y_train)
y_pred = rr.predict(X_test)

print("R-Squared: ", r2_score(y_test,
y_pred))
print("Weights: ", lr.coef_)
print("Constant: ", lr.intercept_)
```

Lasso Regression

```
from sklearn.linear_model import
Lasso
lasso = Lasso()
lasso.fit(X_train, y_train)
y_pred = lasso.predict(X_test)

print("R-Squared: ", r2_score(y_test,
y_pred))
print("Weights: ", lasso.coef_)
print("Constant: ", lasso.intercept_)
```

3)

Logistic_Regression

```
from sklearn.datasets import load_iris
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size = 0.4, random_state = 17)

model = LogisticRegression()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print(classification_report(y_test, y_pred, target_names =
iris.target_names))
print('\nAccuracy: {0:.4f}'.format(accuracy_score(y_test, y_pred)))

from sklearn.metrics import confusion_matrix
C = confusion_matrix(y_test, y_pred)
print(C)
print()
print(C.sum(axis=1))
```

4)

```
from sklearn.datasets import load_iris
from sklearn import tree
from sklearn.metrics import classification_report, accuracy_score
from sklearn.model_selection import train_test_split
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target,
test_size = 0.4)
clf_entropy = tree.DecisionTreeClassifier(random_state = 17, criterion
= 'entropy', min_samples_split=50)
clf = clf_entropy.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print(classification_report(y_test, y_pred, target_names =
iris.target_names))
print('\nAccuracy: {0:.4f}'.format(accuracy_score(y_test, y_pred)))

from sklearn.metrics import confusion_matrix
C = confusion_matrix(y_test, clf_entropy.predict(X_test))
print(C)
print()
print(C.sum(axis=1))

from sklearn.tree import export_graphviz
from io import StringIO
from IPython.display import Image
import pydotplus
import graphviz
import pydot
dot_data = StringIO()
tree.export_graphviz(clf, out_file=dot_data,
feature_names=iris.feature_names, class_names=iris.target_names,
filled=True, rounded=True, special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

5)

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
dataset = pd.read_csv("advertising.csv")
dataset.head()
```

Data Pre-Processing

```
dataset.shape
```

1. Checking for missing values

```
dataset.isna().sum()
```

2. Checking for duplicate rows

```
dataset.duplicated().any()
```

3. Checking for outliers

```
fig, axs = plt.subplots(3, figsize = (5,5))
plt1 = sns.boxplot(dataset['TV'], ax = axs[0])
plt2 = sns.boxplot(dataset['Newspaper'], ax = axs[1])
plt3 = sns.boxplot(dataset['Radio'], ax = axs[2])
plt.tight_layout()
```

Exploratory Data Analysis

1. Distribution of the target variable

```
sns.distplot(dataset['Sales']);
```

2. How Sales are related with other variables

```
sns.pairplot(dataset, x_vars=['TV', 'Radio', 'Newspaper'], y_vars='Sales',
height=4, aspect=1,kind=’scatter’)
plt.show()
```

3. Heatmap

```
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```

Model Building

1. Simple Linear Regression

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics

#Setting the value for X and Y
x = dataset[['TV']]
y = dataset['Sales']
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 1
slr= LinearRegression()
slr.fit(x_train, y_train)

#Printing the model coefficients
print('Intercept: ', slr.intercept_)
print('Coefficient:', slr.coef_)
print('Regression Equation: Sales = 6.948 + 0.054 * TV')


#Line of best fit
plt.scatter(x_train, y_train)
plt.plot(x_train, 6.948 + 0.054*x_train, 'r')
plt.show()
#Prediction of Test and Training set result
y_pred_slr= slr.predict(x_test)
x_pred_slr= slr.predict(x_train)
print("Prediction for test set: {}".format(y_pred_slr))
slr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_slr})
slr_diff

slr.predict([[56]])

# print the R-squared value for the model
from sklearn.metrics import accuracy_score
print('R squared value of the model: {:.2f}'.format(slr.score(x,y)*100))

meanAbErr = metrics.mean_absolute_error(y_test, y_pred_slr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_slr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_slr))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr)
```

2. Multiple Linear Regression

```
#Setting the value for X and Y
x = dataset[['TV', 'Radio', 'Newspaper']]
y = dataset['Sales']
x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.3, random_state=100)
mlr= LinearRegression()
mlr.fit(x_train, y_train)
#Printing the model coefficients
print(mlr.intercept_)
# pair the feature names with the coefficients
list(zip(x, mlr.coef_))
#Predicting the Test and Train set result
y_pred_mlr= mlr.predict(x_test)
x_pred_mlr= mlr.predict(x_train)
print("Prediction for test set: {}".format(y_pred_mlr))

mlr_diff = pd.DataFrame({'Actual value': y_test, 'Predicted value': y_pred_mlr})

mlr.predict([[56, 55, 67]])

# print the R-squared value for the model
print('R squared value of the model: {:.2f}'.format(mlr.score(x,y)*100))
meanAbErr = metrics.mean_absolute_error(y_test, y_pred_mlr)
meanSqErr = metrics.mean_squared_error(y_test, y_pred_mlr)
rootMeanSqErr = np.sqrt(metrics.mean_squared_error(y_test, y_pred_mlr))
print('Mean Absolute Error:', meanAbErr)
print('Mean Square Error:', meanSqErr)
print('Root Mean Square Error:', rootMeanSqErr )
```

6)

```
#importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
#Import mall dataset
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:,[3,4]].values
dataset.head()

plt.scatter(dataset['Annual Income (k$)'],dataset['Spending Score (1-100)'])
plt.xlim(-180,180)
plt.ylim(-90,90)
plt.show()

from sklearn.cluster import KMeans
wcss = []
for i in range(1,11):
 kmeans = KMeans(n_clusters = i, init='k-means++',max_iter = 300, n_init=10, random_state
= 0)
 kmeans.fit(X)
 wcss.append(kmeans.inertia_)
#Plot the graph to visualize the Elbow Metrhod to find the optimal number of cluster
plt.plot(range(1,11),wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()

kmeans = KMeans(n_clusters = 5, init='k-means++',max_iter = 300, n_init=10, random_state
= 0)
Y_Kmeans = kmeans.fit_predict(X)
#Visualizing the clusters
plt.scatter(X[Y_Kmeans ==0,0],X[Y_Kmeans ==0,1],s=100,c='yellow',label = 'Cluster 1')
plt.scatter(X[Y_Kmeans ==1,0],X[Y_Kmeans ==1,1],s=100,c='blue',label = 'Cluster 2')
plt.scatter(X[Y_Kmeans ==2,0],X[Y_Kmeans ==2,1],s=100,c='green',label = 'Cluster 3')
plt.scatter(X[Y_Kmeans ==3,0],X[Y_Kmeans ==3,1],s=100,c='cyan',label = 'Cluster 4')
plt.scatter(X[Y_Kmeans ==4,0],X[Y_Kmeans ==4,1],s=100,c='magenta',label = 'Cluster 5')
plt.scatter(kmeans.cluster_centers_[:,0], kmeans.cluster_centers_[:,1], s = 300, c = 'red',
label = 'Centroids')
plt.title('Clusters of clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending score (1-100)')
plt.legend()
plt.show()
```

7)

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

dataset = pd.read_csv(r'C:\Users\mldata\Social_Network_Ads.csv')
dataset.head(6)

X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
print(X)
print(y)
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])

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=0)

from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)

y_test

y_pred

from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
ac = accuracy_score(y_test,y_pred)

print(cm)
print(ac)
```