Exploratory data Analysis Lab:

Basic Python Packages:

Numpy:

```
import numpy as np
array = np.array ([1,2,3,4])
array_sum = np.sum (array)
Array_mean = np.mean (array)
print("Array :", array)
print ("Sum :", array_sum)
print (" Mean :", array_mean)

matrix = np.array ([[ 1, 3], [3,4]])
transpose = matrix.T
determinant = np.linalg.det (matrix)
print ("Matrix: \n", matrix)
print ("Transpose :\n", transpose)
print (" Determinant :\n", determinat)
```

Pandas:

```
import pandas as pd
data = {"name":['Alice', 'Bob', 'Charlie'], 'Age': [20, 32, 34]}
df=pd.DataFrame (data)
print (df)
print (df['name'])
print (df.describe())
```

④
```
data = {'Name': ['Alice', 'Bob', 'Charlie'], 'Age': [24, 27, 22],
        'Score': [85, 83, 79]}
df = pd.DataFrame(data)
df['pass'] = df['Score'] > 80
print("DataFrame:\n", df)
print("Average Age:", df['Age'].mean())
```

Matplotlib:

⑤
```
import numpy as np
import matplotlib.pyplot as plt
y1 = [2, 1, 4.5]
y2 = [1, 1.5, 4]
plt.plot(y1)
plt.plot(y2)
plt.legend(["blue", "green"], loc="lower right")
plt.show()
```

⑥
```
import matplotlib.pyplot as plt
x = [0, 1, 2, 4, 7]
y = [0, 1, 4, 9, 18]
plt.plot(x, y, marker='D', label='y = x^2')
plt.title("Simple plot")
plt.xlabel("x-axis")
plt.ylabel("y-axis")
plt.legend()
plt.grid()
plt.show()
```

Seaborn:

7
```
import seabon as sns
import matplotlib.pyplot as plt
tips = sns.load_dataset ("tips")
sns.scatterplot (data = tips, x = "total_bill", y = "tip",
                 hue = "sex")
plt.title ("Tips dataset scatter plot")
plt.show ()
```

8
```
import seaborn as sns
import matplotlib.pyplot as plt
tips = sns.load_dataset ("tips")
sns.boxplot (x = "day", y = "total_bill", data=tips)
plt.title ("Total bill distribution by day")
plt.show ()
```

Scikit-learn:

```
from sklearn.datasets import load_iris
iris = load_iris ()
x = iris.data
y = iris.target
feature_names = iris.feature_names
target_name = iris.target_names
print ("feature_names:", feature_names)
print ("target names:", target_names)
print ("\nType of x is:" type (x))
print ("\n first 5 rows of x:\n", x[:5])
```

(10)

```
from sklearn.dataset import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
form sklearn.metrics import accuracy_score
iris = load_iris()
x,y = iris.data, iris.target
x_train, x_test, y_train, y_test = train_test_split
                                   (x,y, test_size = 0.3)
clf = DecisionTreeClassifier()
clf.fit(x_train, y_train)
y_pred = clf.predict(x_test)
accuracy = accuracy_score (y_test, y_pred)
print("Accuracy of the Decision Tree classifier:",
                                   accuracy)
```

①

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

dataset = pd.read_csv("C:/users/android/Desktop/
                       diabetes.csv")
print(dataset)
```

6. dataset = dataset.drop_duplicates()

dataset.isnull()

7. dataset.sum()

② dataset.head()

③ dataset.tail()

④ dataset.describe()

(7) dataset.info()

(8)
```
# Preprocessing tech.
# data Cleaning
import pandas as pd
from sklearn.impute import SimpleImputer
data = pd.DataFrame({
    'name': ['vanya', 'prithvi', 'siri', 'vanya', None],
    'age': [21, 34, None, 21, 22],
    'purchase amount': [100.5, None, None, 100.5, 50.5],
    'date of Purchase': ['2023/12/01', '2023/12/02', '2025/2/01',
        '2023/12/01', '2023/12/03']})
imputer = SimpleImputer(strategy='mean')
data['age', 'purchase-amount'] = imputer.fit_transform
        (data['date-of-purchase'], errors='coerce')
print(data)
```

9 #data intergation

```
import pandas as pd
data1 = pd.DataFrame({
    'cust id': [1,2,3],
    'name': ['Sam','Tom','Ram'],
    'age': [28,34,29]})
data2 = pd.DataFrame({
    'cust id': [1,3,4],
    'purchase_amount': [100.5,85.3,45.0],
    'purchase_data': ['2023-12-01','2023-12-02','2023-12-03']
    })
merged_data = pd.merge(data1,data2,on='cust_id',how='inner')
print(merged_data)
merged_data = pd.merge(data1,data2,on='cust_id',how='outer')
print(merged_data)
merged_data = pd.merge(data1,data2,on='cust_id',how='right')
print(merged_data)
merged_data = pd.merge(data1,data2,on='cust_id',how='left')
print(merged_data)
```

10 #data transformation

```
from sklearn.preprocessing import StandredScalar, OneHotEncoder
data=pd.DataFrame({'category': ['A','B','A','C','B'],
                   'numerical_column': [10,15,10,20,15]})
scaler = StandardScaler()
data['scaled_numeric_column'] = scaler.fit_transform(data[[
                'category']]),

columns = encoder.get_feature_names_out(['category']))
data = pd.concat([data, encoded_data], axis=1)
print(data)
```

#data reduction

```
from sklearn.decomposition import PCA
from sklearn.features_selection import SelectionKBest, chi2
data=pd.DataFrame({'feature1':[10,20,30,40,50],
                   'feature2':[1,2,3,4,5],
                   'feature3':[100,200,300,400,500],
                   'target':[0,1,0,1,0]})
selector=SelectKBest(chi2,k=2)
selected_features= selector.fit_transform(data[['feature1',
        'feature2','feature3']],data['target'])
Print("selected features(select k Best1:")
print(seleeted_features)
pca= PCA(n_components=2)
pca_data=pca.fit_transform(data[['feature1',feature2',
        'feature3']])
print("PCA reduced data:")
print(pca_data)
```

Data Visualization: Univariant Analysis

① Count plot:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(15,8))
sns.countplot(x="Department", data=df)
plt.title("Count plot of department")
plt.show()
```

Pie-chart:

②

```
plt.figure(figsize=(8,8))
df['department'].value_count().plot.pi
plt.title("Department Distribution")
plt.show()
```

③ Histogram:

```
plt.figure(figsize=(10,5))
sns.histplot(df['Salary'], bins=10, color='blue')
plt.title("Histogram of salary")
plt.show().
```

Histogram with distplot:

```
sns.histplot(df['Salary'], bins=10, kde=True, color='blue')
plt.title("Histogram of salary with KDE (distplot)")
plt.show()
```

Boxplot:

```
plt.figure(figsize=(8,6))
sns.boxplot(y=df['Salary'])
plt.title("Salary Boxplot")
plt.show()
```

Bivariant & Multivariant Analysis:

scatter plot

```
plt.figure(figsize=(8, 6))
sns.scatterplot(x="Experiance", y='salary", hue='Department'
                ,data=df)
plt.title("Experiance Vs Salary")
plt.show()
```

7 Pairplot:

```
sns.pairplot(df, here = 'Department')
plt.show()
```

8 Correlation Heatmap:

```
plt.figure(fig size = (8, 6))
sns.Heatmap(df.Select_dtypes(include = ['number'])
color(), annot = true, map = 'coolmorm')
plt.fitle("Corelation heatmap")
plt.$how()
```

Barplot :

```
plt.figure(figsize=(10,5))
sns.barplot(x='department', y='salary', data=df, estimator=
                                    np.mean)
plt.title("Average salary by department")
plt.show()
```

Box plot :

```
plt.figure(figsize=(10,6))
sns.boxplot(x='department', y='salary')
plt.title("salary distribution Across")
plt.show()
```

Time series Analysis: Line plot

```
df_sorted = df.sort_values('joining year')
plt.figure(figsize=(10,5))
sns.lineplot(x='joining year', y='salary', data=df_sorted
             marker='o')
plt.title("Salary treand over joing year")
plt.show()
```

Data Aggregation, Transformation, plot, table & charts

① import pandas as pd

```
data = {'Name': ['Tom', 'Nick', 'Krish', 'Jack'],
        'Age': [20, 21, 19, 18]}
df = pd.DataFrame(data)
print(df)
```

② import pandas as pd

```
data = {'name': ['jai', 'Princi', 'gaurav', 'anuj'],
        'age': [27, 24, 23, 32],
        'qualification': ['Msc', 'MCA', 'MCA', 'Phd'],
        'address': ['Delhi', 'Kanpur', 'Allahabad', 'Kannauj']}
df = pd.DataFrame(data)
print(df[['Name', 'qualification']])
```

③

```
df = pd.DataFrame({'Column1': ['A', 'B', 'C', 'A', 'C',
                               'C', 'B', 'D', 'D', 'A'],
                   'column 2': [5, 10, 15, 20, 25, 30, 35, 40, 45, 50]})
df = df.groupby('Column 1')['column 2'].apply(list)
df
```

④

```
df = pd.DataFrame({'animal': ['Falcon', 'Falcon', 'Parrot', 'Parrot'],
                   'maxspeed': [380, 370, 24, 26]})
df.groupby(['animal']).mean()
```

⑤
```
import numpy as np
import pandas as pd
df = pd.DataFrame({'key1': ['a', 'A', 'b', 'b', 'a'],
    'key2': ['one', 'two', 'one', 'two', 'one'],
    'data1': np.random.rand(5),
    'data2': np.random.rand(5)})
df.groupby(['key1', 'key2']).size()
```

⑥
```
#groping index levels
columns = pd.MultiIndex.from_arrays([
 ['US', 'US', 'US', 'JP', 'JP'], [1,3,5,1,3]], names=['city', 'tenor'])
hier_df = pd.DataFrame(np.random.rand(4,5),
        columns = columns)
hier_df
```

⑦
```
hier_df.groupby(level='city', axis=1).count()
```

⑧
```
#data Integration.

df = pd.DataFrame({'key1': ['a', 'A', 'b', 'b', 'a'],
    'key2': ['one', 'two', 'one', 'two', 'one'],
    'data1': [np.random.rand(5),
    'data2': np.random.rand(5)})
```

```
df.describe()
```

#Aggregation functions

```
df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9],[np.nan],
                   np.nan, np.nan]])
columns = ['A','B','C']
df.agg(['sum','min'], 'B':['min','max']})
```

#aggregate over the columns

```
df = pd.DataFrame({'A':range(3), 'B':range(1,4)})
df.
df = pd.DataFrame({'A':range(3), 'B':range(0,4)})
df
```

```
s = pd.series(range(3))
s.transform([np.sqrt, np.exp])
```

(12) df = pd.DataFrame({
"Date": ["2015-05-08", "2017-05-07", "2015-05-06",
"2015-05-05", "2015-05-08", "2017-05-07",
"2017-05-06", "2015-05-05"],
"Data": [5, 8, 6, 1, 50, 100, 60, 120]})

df

(13) df.groupby ['Date']['Data'].transform(sum).

(14) import seaborn as sns
tips = sns.load_dataset ('tips')
tips.

(15) ~~import seaborn as sns~~

~~tips = sns.load_Dataset('tips')~~

~~tips~~ pd.pivot_table(tips, index="sex", values
= ['tips'])

pd.pivot_table(tips, index="sex", values=['tips'],
aggfunc="sum")

pd.pivot_table(tips, index="sex", values=["tip"),
columns="time")

(16) pd.pivot_table(tips, index=["sex"], values=("total_bill",
"tip"], columns=["time,"smoker"])

pd.crosstab(tips["sex"], tips["time"])

pd.Crosstab(tip["Sex"], tips["time"], normalize='columns')

(17) pd.crosstab(tips["sex"], tips["time"], normalize="all")

pd.crosstab(tips("sex"), tips("time"), value=tips
["tip"], aggfun=("mean")

pd.crosstab(tips("sex"], tips["time"], values=tips["tip"],
aggfun = "mean", rownames=["Gender"],
columns=["Time"])

pd. crosstab(tips("sex"), [tips["time"], tips["smoker"]
D

(18) pd.crosstab([tabs["sex"], tips["day"]], [tips["time"],
tips["smoker"]])

(B) pd.crosstab(tips["day"], tips["sex"], normalize='index').plot.bar(stacked=False)

(20)
```
import pandas as pd
student = {'name': ['Vaish', 'Sham', 'anj', 'anu'],
           'math': [99, 98, 76, 63],
           'ML': [88, 77, 89, 89],
           'total_score': [155, 233, 266, 224],
           'Grating': ['A', 'B', 'A', 'B'],
           'age': [20, 21, 20, 20]}
df = pd.DataFrame(Student)
df
print(df[['name', 'grading']])
df = df.groupby('name')['total_score'].apply(list)
```

Linear Regression:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression,
                Ridge, Lasso, ElasticNet
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_Error

np.random.seed(42)
x = 2*np.random.rand(100,1)
y = 4+3*x+np.random.rand(100,1)

x_train, x_test, y_train, y_test = train_test_split
                (x,y, test_size=0.2, random_state=42)

lin_reg = LinearRegression()
lin_reg.fit(x_train, y_train)
y_pred = lin_reg.predict(x_test)
print("Simple Linear Regression MSE:", mean_squared_
                Error(y_test, y_pred))

plt.figure(figsize=(8,6))
plt.scatter(x,y, color='blue', label='Actual data')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

```
x_multi = np.c_[x, x**2]
x_train_m, x_test_m, y_train_m, y_test_m = train_test_split(x_multi, y, test_size=0.2, random_state=42)
lin_reg_multi = LinearRegression()
lin_reg_multi.fit(x_train_m, y_train_m)
y_pred_multi = lin_reg_multi.predict(x_test_m)
print("Multiple Linear Regression MSE", mean_squared_error(y_test_m, y_pred_multi))
plt.figure(figsize=(8,6))
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x_test_m, y_pred_multi, color='green', label='Multiple Linear Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()
poly_features = PolynomialFeatures(degree=2, include_bias=False)
x_poly = poly_features.fit_transform(X)
x_train_p, x_test_p, y_train_p, y_test_p = train_test_split(x_poly, y, test_size=0.2, random_state=42)
poly_reg = LinearRegression()
poly_reg.fit(x_train_p, y_train_p)
y_pred_poly = poly_reg.predict(x_test_p)
print("polynomial Regression MSE", mean_squared_error(y_test_p, y_pred_poly))
```

```
plt.figure(figsize=(8,6))
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x_test_p[:,0], y_pred_poly, color='orange',
         label='Polynomial Regression')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show()

Ridge_reg = Ridge(alpha=1.0)
ridge_reg.fit(x_train_p, y_train_p)
y_pred_ridge = regid_reg.predict(x_test_p)
print("Ridge Regression MSE", mean_squared_error
      (y_test_p, y_pred_ridge))
plt.figure(figsize=(8,6))
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x_test_p[:,0], y_pred_ridge, color='purple',
         label='Ridge-Regression')
plt.xlabel('x')
plt.ylabel('y')
plt.legend()
plt.show().

lasso_reg = Lasso(alpha=0.1)
Lasso_reg.fit = (x train_p, y_train_p)
p_pred_lasso = lasso_reg.predict(x_test_p)
print("Lasso Regression MSE=", mean_squared_error
      (y_test_p, y_pred_lasso))
plt.figure(figsize=(8,10))
```

```
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x_test_p[:,0], y_pred_lasso, color='brown',
         label='Lasso Regression')
plt.xlabel('X')
plt.ylabel('y')
plt.legend()
plt.show()

Elastic_net = ElasticNet(alpha=0.1, l1_ratio=0.5)
elastic_net.fit(x_train_p, y_train_p)
y_pred_elastic = Elastic_net.predict(x_test_p)
print("Elastic Net Regression MSE:", Mean_Squeared_
      error(y_test_p, y_pred_elastic))
plt.figure(figsize=(8,6))
plt.scatter(x, y, color='blue', label='Actual Data')
plt.plot(x_test_p[:,0], y_pred_elastic, color='pink',
    label='Elastic Net Regression')
plt.xLabel('x')
plt.ylabel('y')
plt.legend()
plt.show()
```

Logistic Regression:

27/2/25

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.dataset import load_iris
data = load_iris()
x, y = data.data, data.target
x_train, x_test, y_train, y_test = train_test_split
                    (x, y, test_size=0.2, random_state=42)
scaler = StandardScaler()
x_train = scaler.fit.transform (x_train)
x_test = scaler.transform (x_test)

# Binary Logistic
binary_y_train = (y_train == 0).astype(int)
binary_y_test = (y_test == 0).astype(int)
log_reg_binary = LogisticRegression()
log_reg_binary.fit(x_train, binary_y_train)
preds_binary = log_reg_binary.predict (x_test)
print("Binary Logistic Regression accuracy:",
        accuracy_score (binary_y_test, preds_binary))

# Multiclass logistic
log_reg_over = LogisticRegression (multi_class='ovr',
                    solver='lbfgs', max_iter=200)
log_reg_over.fit (x_train, y_train)
preds_over = log_reg_over.predict (x_test)
print ("One-vs-Rest logistic Regression accuracy:",
        accuracy_score (y_test, preds))
```

```
# Logistic regression softmax
log_reg_softmax = LogisticRegression(multi_class
        ='over", solver = 'lbfgs', max_iter = 200)

log_reg_softmax.fit(x_train, y_train)
preds_softmax = log_reg_softmax.predict(x_test)

print("softmax Logistic Regression accuracy:",
        accuracy_score(y_test, preds_softmax))


#L1, Ld°, Elastic Net.
log_reg_l1 = LogisticRegression(penalty='l1', solver=
        'liblinear', max_iter=200)
log_reg_l1.fit(x_train, y_train)
preds_l1 = log_reg_l1.predict(x_test)          preds_l1))
print("L1 logistic accuracy:", accuracy_score(y_test, ^

log_reg_l2 = LogisticRegression(penalty='l2', max_iter=200)
log_reg_l2.fit(x_train, y_train)
preds_l2 = log_reg_l2.predict(x_test)
print("L2 logistic accuracy:", accuracy_score(y_test, preds_l2))
log_reg_elastic = LogisticRegression(penalty = 'Elastic net',
        solver = 'saga', l1_ratio = 0.5, max_iter=200)
log_reg_elastic.fit(x_train, y_train)
preds_elastic = log_reg_elastic.predict(x_test)
print("Elastic Net logistic Regression Accuracy",
        accuracy_score(y_test, preds_elastic))

# bar plot to compair
regression_types = ['Multiclass', 'multiclass with softmax',
    'regularization', 'L1', 'L2', 'elasticNet']
accuracy = [0.75, 0.88, 0.91, 0.94, 0.96]
plt.bar(Regression_types, accuracy, color = 'skyblue')
plt.title('Regresion types Vs Accuracies')
plt.show()
```

K-Nearest Neighbor Classifier:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.datasets import make_classification.

X, y = make_classification(n_samples = 300, n_features=5,
        n_classes=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                    test_size=0.2, random_state=42)

scaler = StandredScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
k = 7
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train, y_train)
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"accuracy: {accuracy:.2f}")
print("Classification Report:")
print(Classification_report(y_test, y_pred))
```

Decision Tree:

```
from sklearn.dataset import load_iris
from sklearn.tree import DecisionTreeClassifier,
                         plot_tree
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

X, y = iris.data , iris.target

X_train, X_test, y_train, y_test = train_test_split(
                X, y, test_size=0.2, random_state=42)

clf = DecisionClassifier(criterion='gini', max_depth=3,
                         random_state=42)

clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:2f}")

plt.figure(figsize=(12,8))

plot_tree(clf, feature_name=iris.feature_names,
    class_names=iris.target_names, filled=True)

plt.title("Decision tree Visualization")

plt.show()
```

SVM:

```
from sklearn import svm, dataset
import sklearn.model_selection as model_selection
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

iris = load_iris()
x = iris.data[:, :2]
y = iris.target

x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y, train_size=0.80, test_size=0.2, random_state=101)

rbf = svm.SVC(kernal='rbf', gamma=0.5, C=0.1).fit(x_train, y=train)

poly = svm.SVC(kernal='poly', degree=3, c=1).fit(x_train, y_train)

poly_pred = poly.predict(x_test)
rbf_pred = rbf.predict(x_test)
poly_accuracy = accuracy_score(y_test, poly_test)
poly_f1 = f1_score(y_test, poly_pred, average='weighted')
print("Accuracy (Polynomial kernal):", "%.2f" %.(poly_f1*100)
rbf_accuracy = accuracy_score(y_test, rbf_pred)
rbf_f1 = f1_score(y_test, poly_pred, average='weighted')
print("Accuracy (RBF kernal):", "%.2f" %.(rbf_accuracy*100))
print("f1 (RBF kernal):", "%.2f" %.(rbf_f1*100))
```