In [111…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, accuracy_score
import tensorflow.keras
from keras.models import Sequential
from keras.layers import Dense, LeakyReLU
%matplotlib inline
```

In [112…

```
df = pd.read_csv('facebook_ads.csv', encoding='ISO-8859-1')
df
```

Out[112…

	Names	emails	Country	Time Spent on Site	Salary	Clicked
0	Martina Avila	cubilia.Curae.Phasellus@quisaccumsanconvallis.edu	Bulgaria	25.649648	55330.06006	0
1	Harlan Barnes	eu.dolor@diam.co.uk	Belize	32.456107	79049.07674	1
2	Naomi Rodriquez	vulputate.mauris.sagittis@ametconsectetueradip...	Algeria	20.945978	41098.60826	0
3	Jade Cunningham	malesuada@dignissim.com	Cook Islands	54.039325	37143.35536	1
4	Cedric Leach	felis.ullamcorper.viverra@egetmollislectus.net	Brazil	34.249729	37355.11276	0
...	...	...	...	...	...	...
494	Rigel	egestas.blandit.Nam@semvitaealiquam.com	Sao Tome and Principe	19.222746	44969.13495	0
495	Walter	ligula@Cumsociis.ca	Nepal	22.665662	41686.20425	0
496	Vanna	Cum.sociis.natoque@Sedmolestie.edu	Zimbabwe	35.320239	23989.80864	0
497	Pearl	penatibus.et@massanonante.com	Philippines	26.539170	31708.57054	0
498	Nell	Quisque.varius@arcuVivamussit.net	Botswana	32.386148	74331.35442	1

499 rows × 6 columns

Exeploration Datasets

In [113…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499 entries, 0 to 498
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Names               499 non-null    object 
 1   emails              499 non-null    object 
 2   Country             499 non-null    object 
 3   Time Spent on Site  499 non-null    float64
 4   Salary              499 non-null    float64
 5   Clicked             499 non-null    int64  
dtypes: float64(2), int64(1), object(3)
memory usage: 23.5+ KB
```

In [114…

```
df.duplicated().sum()
```

Out[114…

```
0
```

In [115…

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

Out[115…

```
Names                 0
emails                0
Country               0
Time Spent on Site    0
Salary                0
Clicked               0
dtype: int64
```

In [116…

```
df.describe().round(2).T
```

Out[116…

	count	mean	std	min	25%	50%	75%	max
Time Spent on Site	499.0	32.92	9.10	5.0	26.43	33.20	39.11	60.0
Salary	499.0	52896.99	18989.18	20.0	38888.12	52840.91	65837.29	100000.0
Clicked	499.0	0.50	0.50	0.0	0.00	1.00	1.00	1.0

Clean Datasets

In [117…
```
def clean_outliers(df, cols):

    for col in cols:
        # Calculate IQR
        q1 = df[col].quantile(0.25)
        q3 = df[col].quantile(0.75)
        iqr = q3 - q1

        # Define outlier bounds
        lower_bound = q1 - 1.5 * iqr
        upper_bound = q3 + 1.5 * iqr

        # Remove outliers
        df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

    return df
```

In [118…
```
df= clean_outliers(df,df[['Time Spent on Site', 'Salary']])
```

In [119…
```
df.drop(columns=['Names','emails', 'Country'], inplace=True)
```

In [120…
```
df
```

Out[120…

	Time Spent on Site	Salary	Clicked
0	25.649648	55330.06006	0
1	32.456107	79049.07674	1
2	20.945978	41098.60826	0
3	54.039325	37143.35536	1
4	34.249729	37355.11276	0
...	...	...	...
494	19.222746	44969.13495	0
495	22.665662	41686.20425	0
496	35.320239	23989.80864	0
497	26.539170	31708.57054	0
498	32.386148	74331.35442	1

497 rows × 3 columns

Machine Learning:

In [121…

```
scaler = StandardScaler()
```

In [122…

```
X = df[['Time Spent on Site', 'Salary']]
y= df['Clicked']
```

In [123…

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

In [124…

```
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

1- Logistic Regression:

In [125…
```
lr = LogisticRegression()
scores_logistic_regression = cross_val_score(lr, X, y, cv=5, scoring='accuracy')
print(f'Accuracy Score with Logistic Regression = {scores_logistic_regression.mean().round(2)}')
```

```
Accuracy Score with Logistic Regression = 0.91
```

2- Support Vector Classifier:

In [126…
```
svc = SVC()
scores_svc = cross_val_score(svc, X, y, cv=5, scoring='accuracy')
print(f'Accuracy Score with Support Vector Classifier = {scores_svc.mean().round(2)}')
```

```
Accuracy Score with Support Vector Classifier = 0.81
```

3- Random Forest Classifier:

In [127…
```
rfc = RandomForestClassifier()
scores_Random_Forest_Classifier = cross_val_score(rfc, X, y, cv=5, scoring='accuracy')
print(f'Accuracy Score with scores Random Forest Classifier = {scores_Random_Forest_Classifier.mean().round(2)}')
```

```
Accuracy Score with scores Random Forest Classifier = 0.89
```

Deep Learning:

In [128…
```
ann = Sequential()
ann.add(Dense(units=50)) # 1st Hidden Layer
ann.add(LeakyReLU())
ann.add(Dense(units=50, activation='relu')) # 2nd Hidden Layer
ann.add(Dense(units=1, activation='sigmoid')) # Output Layer
```

In [129…
```
ann.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

In [130…
```
ann.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

```
Epoch 1/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.5753 - loss: 0.6388
Epoch 2/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8747 - loss: 0.5088
Epoch 3/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8980 - loss: 0.4174
Epoch 4/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8838 - loss: 0.3823
Epoch 5/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8794 - loss: 0.3354
Epoch 6/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8896 - loss: 0.3117
Epoch 7/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9064 - loss: 0.2806
Epoch 8/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9009 - loss: 0.2545
Epoch 9/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9071 - loss: 0.2616
Epoch 10/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8925 - loss: 0.2774
Epoch 11/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9079 - loss: 0.2686
Epoch 12/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9069 - loss: 0.2770
Epoch 13/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9263 - loss: 0.2440
Epoch 14/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9206 - loss: 0.2511
Epoch 15/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8962 - loss: 0.2736
Epoch 16/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9219 - loss: 0.2203
Epoch 17/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9286 - loss: 0.2391
Epoch 18/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9314 - loss: 0.2359
Epoch 19/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9054 - loss: 0.2440
Epoch 20/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9218 - loss: 0.2437
Epoch 21/100
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8696 - loss: 0.3415
Epoch 22/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9162 - loss: 0.2485
Epoch 23/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9266 - loss: 0.2176
Epoch 24/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9141 - loss: 0.2556
Epoch 25/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9057 - loss: 0.2855
Epoch 26/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9036 - loss: 0.2676
Epoch 27/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9236 - loss: 0.2483
Epoch 28/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9283 - loss: 0.2307
Epoch 29/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9055 - loss: 0.2609
Epoch 30/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9195 - loss: 0.2588
Epoch 31/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9122 - loss: 0.2502
Epoch 32/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9103 - loss: 0.2454
Epoch 33/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9054 - loss: 0.2689
Epoch 34/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9098 - loss: 0.2568
Epoch 35/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9079 - loss: 0.2467
Epoch 36/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8975 - loss: 0.2738
Epoch 37/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9165 - loss: 0.2187
Epoch 38/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9004 - loss: 0.2558
Epoch 39/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8950 - loss: 0.2814
Epoch 40/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8860 - loss: 0.2760
Epoch 41/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8930 - loss: 0.2737
```

```
Epoch 42/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8995 - loss: 0.2693
Epoch 43/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9063 - loss: 0.2561
Epoch 44/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9008 - loss: 0.2630
Epoch 45/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9200 - loss: 0.2385
Epoch 46/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9022 - loss: 0.2409
Epoch 47/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8923 - loss: 0.2755
Epoch 48/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9284 - loss: 0.2229
Epoch 49/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8780 - loss: 0.2965
Epoch 50/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9272 - loss: 0.2282
Epoch 51/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9065 - loss: 0.2433
Epoch 52/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9273 - loss: 0.2290
Epoch 53/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9317 - loss: 0.2187
Epoch 54/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9059 - loss: 0.2373
Epoch 55/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8974 - loss: 0.2712
Epoch 56/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8942 - loss: 0.2613
Epoch 57/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9081 - loss: 0.2502
Epoch 58/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8907 - loss: 0.2556
Epoch 59/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9094 - loss: 0.2387
Epoch 60/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9294 - loss: 0.2085
Epoch 61/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9143 - loss: 0.2289
Epoch 62/100
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - accuracy: 0.9141 - loss: 0.2238
Epoch 63/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8956 - loss: 0.2682
Epoch 64/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9025 - loss: 0.2527
Epoch 65/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9160 - loss: 0.2361
Epoch 66/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9016 - loss: 0.2486
Epoch 67/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8892 - loss: 0.2830
Epoch 68/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9109 - loss: 0.2428
Epoch 69/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8982 - loss: 0.2572
Epoch 70/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8893 - loss: 0.2548
Epoch 71/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9045 - loss: 0.2466
Epoch 72/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9138 - loss: 0.2248
Epoch 73/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8871 - loss: 0.2831
Epoch 74/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9088 - loss: 0.2550
Epoch 75/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9157 - loss: 0.2087
Epoch 76/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9042 - loss: 0.2622
Epoch 77/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8938 - loss: 0.2574
Epoch 78/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9004 - loss: 0.2468
Epoch 79/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8915 - loss: 0.2834
Epoch 80/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8978 - loss: 0.2603
Epoch 81/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.8978 - loss: 0.2333
Epoch 82/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9060 - loss: 0.2570
```

```
Epoch 83/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9144 - loss: 0.2235
Epoch 84/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9090 - loss: 0.2232
Epoch 85/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9122 - loss: 0.2272
Epoch 86/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8940 - loss: 0.2712
Epoch 87/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8780 - loss: 0.3122
Epoch 88/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9033 - loss: 0.2393
Epoch 89/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8928 - loss: 0.2576
Epoch 90/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9052 - loss: 0.2412
Epoch 91/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8752 - loss: 0.3056
Epoch 92/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9043 - loss: 0.2590
Epoch 93/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9143 - loss: 0.2419
Epoch 94/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8981 - loss: 0.2559
Epoch 95/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9111 - loss: 0.2335
Epoch 96/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9109 - loss: 0.2326
Epoch 97/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9024 - loss: 0.2505
Epoch 98/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8919 - loss: 0.2682
Epoch 99/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9111 - loss: 0.2184
Epoch 100/100
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8941 - loss: 0.2434
```

Out[130…

```
<keras.src.callbacks.history.History at 0x220f56d9a60>
```

In [131…

```
ann.history.history.keys()
```

Out[131…

```
dict_keys(['accuracy', 'loss'])
```

In [132…

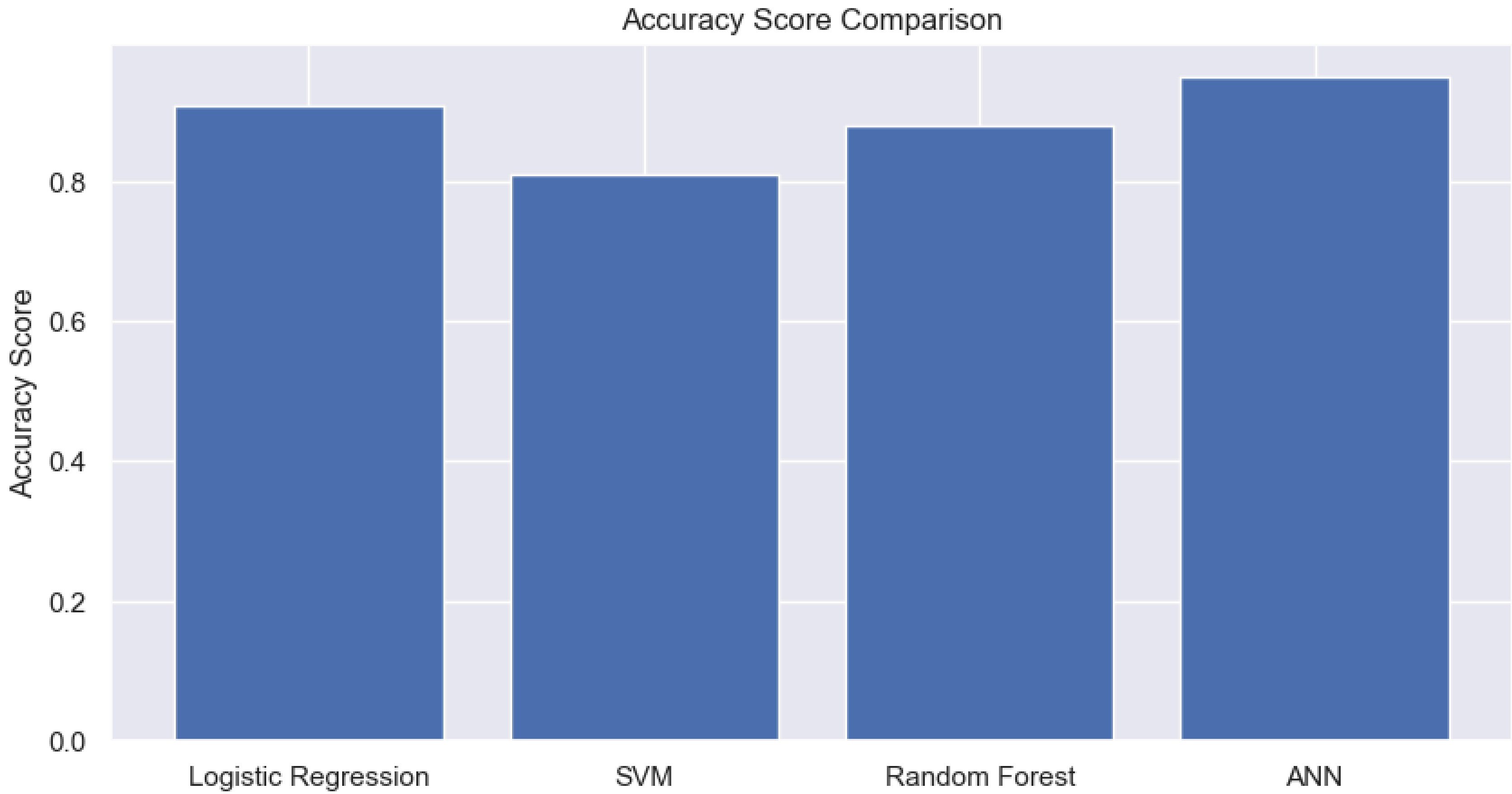
```
y_pred = ann.predict(X_test)
y_pred = (y_pred > 0.5).astype(int)  # Convert probabilities to binary class labels
score_ann= accuracy_score(y_test, y_pred)
print(f'Accuracy Score with ANN = {score_ann}')
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step
Accuracy Score with ANN = 0.95
```

Accuracy score Comparison:

In [133…

```
plt.figure(figsize=(10,5))
models = ['Logistic Regression','SVM', 'Random Forest', 'ANN']
scores = [0.91, 0.81, 0.88, 0.95]
plt.bar(models,scores)
plt.title('Accuracy Score Comparison')
plt.ylabel('Accuracy Score')
plt.show()
```



Summary:

- Exeploration Dataset:

Conducted thorough data cleaning, handled missing values, and detected outliers using the IQR method.

- Machine Learning Models:

Built and evaluated Logistic Regression, Support Vector Machine (SVM), and Random Forest models using cross-validation.

- Deep Learning Model (ANN):

Developed an Artificial Neural Network that achieved the highest performance among all models.

- Key Insights:
  - ANN outperformed all traditional models with higher accuracy.
  - Proper data preprocessing and feature scaling significantly impacted model performance.
  - Cross-validation provided a realistic evaluation compared to a simple train/test split.

Thank you