

```
import pandas as pd
```

## 1. Reading the DataSet

```
data = pd.read_csv("/content/Churn_Modelling.csv")
```

## 2. Display the number of rows and columns of the data

```
data.shape
```

```
(10000, 14)
```

## 3. Display Top 5 rows of the data

```
data.head()
```

```
(10000, 14)
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts
0	1	15634602	Hargrave	619	France	Female	42	2	0.00	
1	2	15647311	Hill	608	Spain	Female	41	1	83807.86	
2	3	15619304	Onio	502	France	Female	42	8	159660.80	
3	4	15701354	Boni	699	France	Female	39	1	0.00	
4	5	15737888	Mitchell	850	Spain	Female	43	2	125510.82	

Next steps:

[Generate code with data](#)

[View recommended plots](#)
[New interactive sheet](#)

## 4. Get information of the dataset such as Column datatype, count of rows and columns and memory required

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 14 columns):
#   Column                Non-Null Count  Dtype
---  -
0   RowNumber              10000 non-null  int64
1   CustomerId             10000 non-null  int64
2   Surname                10000 non-null  object
3   CreditScore            10000 non-null  int64
4   Geography              10000 non-null  object
5   Gender                 10000 non-null  object
6   Age                    10000 non-null  int64
7   Tenure                 10000 non-null  int64
8   Balance                10000 non-null  float64
9   NumOfProducts          10000 non-null  int64
10  HasCrCard              10000 non-null  int64
11  IsActiveMember         10000 non-null  int64
12  EstimatedSalary         10000 non-null  float64
13  Exited                  10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

## 5. Display the Null value count in each column

```
data.isnull().sum()
```



	0
<b>RowNumber</b>	0
<b>CustomerId</b>	0
<b>Surname</b>	0
<b>CreditScore</b>	0
<b>Geography</b>	0
<b>Gender</b>	0
<b>Age</b>	0
<b>Tenure</b>	0
<b>Balance</b>	0
<b>NumOfProducts</b>	0
<b>HasCrCard</b>	0
<b>IsActiveMember</b>	0
<b>EstimatedSalary</b>	0
<b>Exited</b>	0

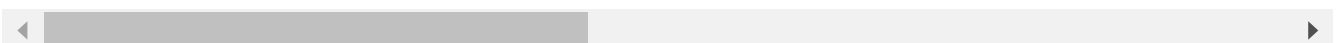


## 6. Get Overall Statistics in the Dataset

```
data.describe(include='all')
```



	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure
<b>count</b>	10000.000000	1.000000e+04	10000	10000.000000	10000	10000	10000.000000	10000.000000
<b>unique</b>	NaN	NaN	2932	NaN	3	2	NaN	NaN
<b>top</b>	NaN	NaN	Smith	NaN	France	Male	NaN	NaN
<b>freq</b>	NaN	NaN	32	NaN	5014	5457	NaN	NaN
<b>mean</b>	5000.500000	1.569094e+07	NaN	650.528800	NaN	NaN	38.921800	5.012800
<b>std</b>	2886.89568	7.193619e+04	NaN	96.653299	NaN	NaN	10.487806	2.892174
<b>min</b>	1.000000	1.556570e+07	NaN	350.000000	NaN	NaN	18.000000	0.000000
<b>25%</b>	2500.750000	1.562853e+07	NaN	584.000000	NaN	NaN	32.000000	3.000000
<b>50%</b>	5000.500000	1.569074e+07	NaN	652.000000	NaN	NaN	37.000000	5.000000
<b>75%</b>	7500.250000	1.575323e+07	NaN	718.000000	NaN	NaN	44.000000	7.000000
<b>max</b>	10000.000000	1.581569e+07	NaN	850.000000	NaN	NaN	92.000000	10.000000



## 7. Display the Column names of the dataset

```
data.columns
```



```
Index(['RowNumber', 'CustomerId', 'Surname', 'CreditScore', 'Geography',  
      'Gender', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
```

```
'IsActiveMember', 'EstimatedSalary', 'Exited'],
dtype='object')
```

## 8. Dropping irrelevant columns from the data to predict the Churn

```
data = data.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)
data.head()
```

	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Est
0	619	France	Female	42	2	0.00	1	1	1	
1	608	Spain	Female	41	1	83807.86	1	0	1	
2	502	France	Female	42	8	159660.80	3	1	0	
3	699	France	Female	39	1	0.00	2	0	0	
4	850	Spain	Female	43	2	125510.82	1	1	1	

Next steps:

[Generate code with data](#)[View recommended plots](#)[New interactive sheet](#)

## 9. Display the unique values in each column

```
# Display unique values in each column
for column in data.columns:
    unique_values = data[column].unique()
    print(f"Unique values in {column}: {unique_values}")
```

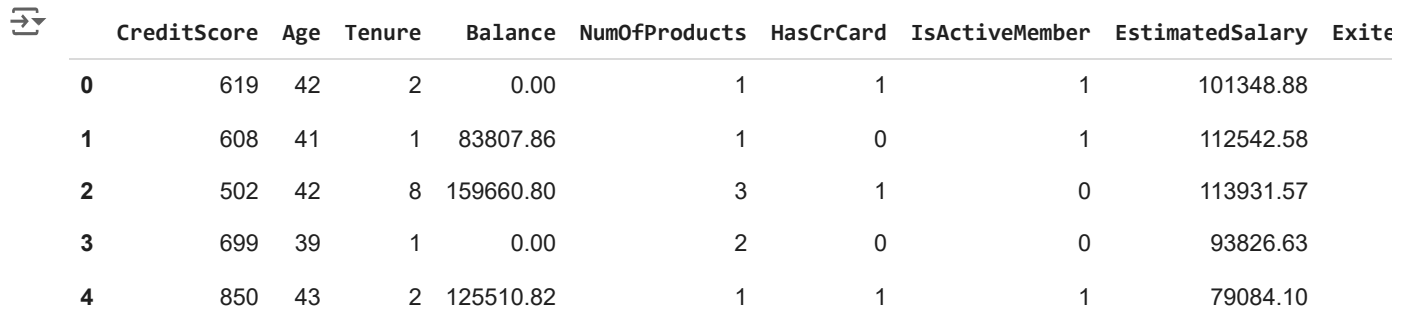
```
Unique values in CreditScore: [619 608 502 699 850 645 822 376 501 684 528 497 476 549 635 616 653 587
726 732 636 510 669 846 577 756 571 574 411 591 533 553 520 722 475 490
804 582 472 465 556 834 660 776 829 637 550 698 585 788 655 601 656 725
511 614 742 687 555 603 751 581 735 661 675 738 813 657 604 519 664 678
757 416 665 777 543 506 493 652 750 729 646 647 808 524 769 730 515 773
814 710 413 623 670 622 785 605 479 685 538 562 721 628 668 828 674 625
432 770 758 795 686 789 589 461 584 579 663 682 793 691 485 650 754 535
716 539 706 586 631 717 800 683 704 615 667 484 480 578 512 606 597 778
514 525 715 580 807 521 759 516 711 618 643 671 689 620 676 572 695 592
567 694 547 594 673 610 767 763 712 703 662 659 523 772 545 634 739 771
681 544 696 766 727 693 557 531 498 651 791 733 811 707 714 782 775 799
602 744 588 747 583 627 731 629 438 642 806 474 559 429 680 749 734 644
626 649 805 718 840 630 654 762 568 613 522 737 648 443 640 540 460 593
801 611 802 745 483 690 492 709 705 560 752 701 537 487 596 702 486 724
548 464 790 534 748 494 590 468 509 818 816 536 753 774 621 569 658 798
641 542 692 639 765 570 638 599 632 779 527 564 833 504 842 508 417 598
741 607 761 848 546 439 755 760 526 713 700 666 566 495 688 612 477 427
839 819 720 459 503 624 529 563 482 796 445 746 786 554 672 787 499 844
450 815 838 803 736 633 600 679 517 792 743 488 421 841 708 507 505 456
435 561 518 565 728 784 552 609 764 697 723 551 444 719 496 541 830 812
677 420 595 617 809 500 826 434 513 478 797 363 399 463 780 452 575 837
794 824 428 823 781 849 489 431 457 768 831 359 820 573 576 558 817 449
440 415 821 530 350 446 425 740 481 783 358 845 451 458 469 423 404 836
473 835 466 491 351 827 843 365 532 414 453 471 401 810 832 470 447 422
825 430 436 426 408 847 418 437 410 454 407 455 462 386 405 383 395 467
433 442 424 448 441 367 412 382 373 419]
Unique values in Geography: ['France' 'Spain' 'Germany']
Unique values in Gender: ['Female' 'Male']
Unique values in Age: [42 41 39 43 44 50 29 27 31 24 34 25 35 45 58 32 38 46 36 33 40 51 61 49
37 19 66 56 26 21 55 75 22 30 28 65 48 52 57 73 47 54 72 20 67 79 62 53
80 59 68 23 60 70 63 64 18 82 69 74 71 76 77 88 85 84 78 81 92 83]
Unique values in Tenure: [ 2  1  8  7  4  6  3 10  5  9  0]
Unique values in Balance: [ 0. 83807.86 159660.8 ... 57369.61 75075.31 130142.79]
Unique values in NumOfProducts: [1 3 2 4]
Unique values in HasCrCard: [1 0]
Unique values in IsActiveMember: [1 0]
```

```
Unique values in EstimatedSalary: [101348.88 112542.58 113931.57 ... 42085.58 92888.52 38190.78]  
Unique values in Exited: [1 0]
```

## 10. Encoding Categorical Data

```
data = pd.get_dummies(data, drop_first = True)
```

```
data.head()
```



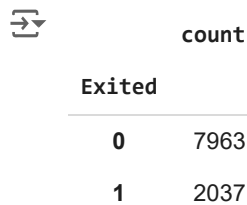
	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
0	619	42	2	0.00	1	1	1	101348.88	
1	608	41	1	83807.86	1	0	1	112542.58	
2	502	42	8	159660.80	3	1	0	113931.57	
3	699	39	1	0.00	2	0	0	93826.63	
4	850	43	2	125510.82	1	1	1	79084.10	

Next steps:

[Generate code with data](#)[View recommended plots](#)[New interactive sheet](#)

## 11. Check the Data Balance

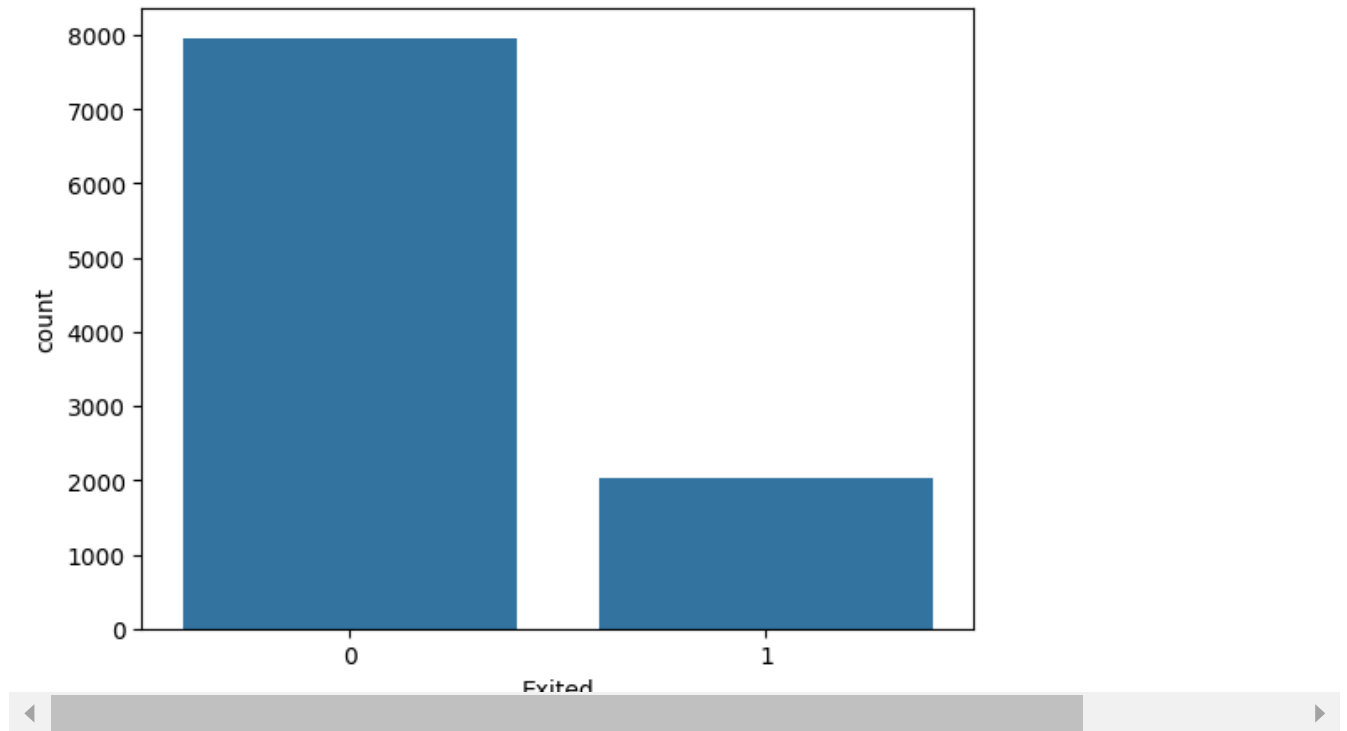
```
data['Exited'].value_counts()
```



	count
Exited	
0	7963
1	2037

```
import seaborn as sns  
sns.countplot(data=data, x='Exited')
```

↩ <Axes: xlabel='Exited', ylabel='count'>



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

## 12. Handling the Data Imbalance

```
from imblearn.over_sampling import SMOTE
X_res, y_res = SMOTE().fit_resample(X,y)
y_res.value_counts()
```

↩ /usr/local/lib/python3.10/dist-packages/sklearn/base.py:474: FutureWarning: `BaseEstimator.\_validate\_data` warnings.warn(  
 /usr/local/lib/python3.10/dist-packages/sklearn/utils/\_tags.py:354: FutureWarning: The SMOTE or classes warnings.warn(  
 count

Exited

1 7963

0 7963

dtype: int64

## 13. Splitting the Data into Training and Testing set

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_res,y_res, test_size=0.2, random_state = 42)
```

## 14. Feature Scaling

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

## 15. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
log.fit(X_train, y_train)
```



▼ LogisticRegression ⓘ ?  
LogisticRegression()

```
y_pred1 = log.predict(X_test)
```

```
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, precision_score, recall_score
```

```
accuracy_score(y_test, y_pred1)
```



0.7849968612680477

```
precision_score(y_test, y_pred1)
```



0.7702366127023661

```
recall_score(y_test, y_pred1)
```



0.7965228589826143

```
f1_score(y_test, y_pred1)
```



0.7831592276036721

## 16. SVC

```
from sklearn import svm
svm = svm.SVC()
svm.fit(X_train, y_train)
```



▼ SVC ⓘ ?  
SVC()

```
y_pred2 = svm.predict(X_test)
```

```
accuracy_score(y_test, y_pred2)
```



0.8452605147520402

```
precision_score(y_test, y_pred2)
```



0.8455019556714471

## 17. KNeighbors Classifier

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
```

```
knn.fit(X_train, y_train)
```



▼ KNeighborsClassifier ⓘ ?  
KNeighborsClassifier()

```
y_pred3 = knn.predict(X_test)
```

```
accuracy_score(y_test, y_pred3)
```



```
0.8207784055241683
```

```
precision_score(y_test, y_pred3)
```



```
0.8023399014778325
```

```
recall_score(y_test, y_pred3)
```



```
0.8390212491951062
```

## 18. Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
y_pred4 = dt.predict(X_test)
```

```
accuracy_score(y_test, y_pred4)
```



```
0.8013182674199624
```

```
precision_score(y_test, y_pred4)
```



```
0.7777777777777778
```

## 19. Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred5 = rf.predict(X_test)
accuracy_score(y_test, y_pred5)
```



```
0.8757062146892656
```

🔗 Generate

a slider using jupyter widgets



Close

```
X_train.shape
```



```
(12740, 11)
```

```
precision_score(y_test, y_pred5)
```



```
0.8686224489795918
```

```
recall_score(y_test, y_pred5)
```

0.8770122343850612

## 20. Gradient Boosting Classifier

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(X_train, y_train)
y_pred6 = gb.predict(X_test)
accuracy_score(y_test, y_pred6)
```

0.8483992467043314

```
precision_score(y_test, y_pred6)
```

0.842948717948718

## 21. To Display the final result in each ML model

```
final_data = pd.DataFrame({'Models': ['LR', 'SVC', 'KNN', 'DT', 'RF', 'GBC'], 'ACC': [accuracy_score(y_test,
```

```
final_data
```

	Models	ACC	PRE	
0	LR	0.784997	0.770237	
1	SVC	0.845261	0.845502	
2	KNN	0.820778	0.802340	
3	DT	0.801318	0.777778	
4	RF	0.875392	0.868622	
5	GBC	0.848399	0.842949	

Next steps:

[Generate code with final\\_data](#)



[View recommended plots](#)

[New interactive sheet](#)

```
import seaborn as sns
```

```
import matplotlib.pyplot as plt
```

```
# Create a bar plot
```

```
sns.barplot(x=final_data['Models'], y=final_data['ACC'])
```

```
# Add titles and labels
```

```
plt.title('Model Accuracy')
```

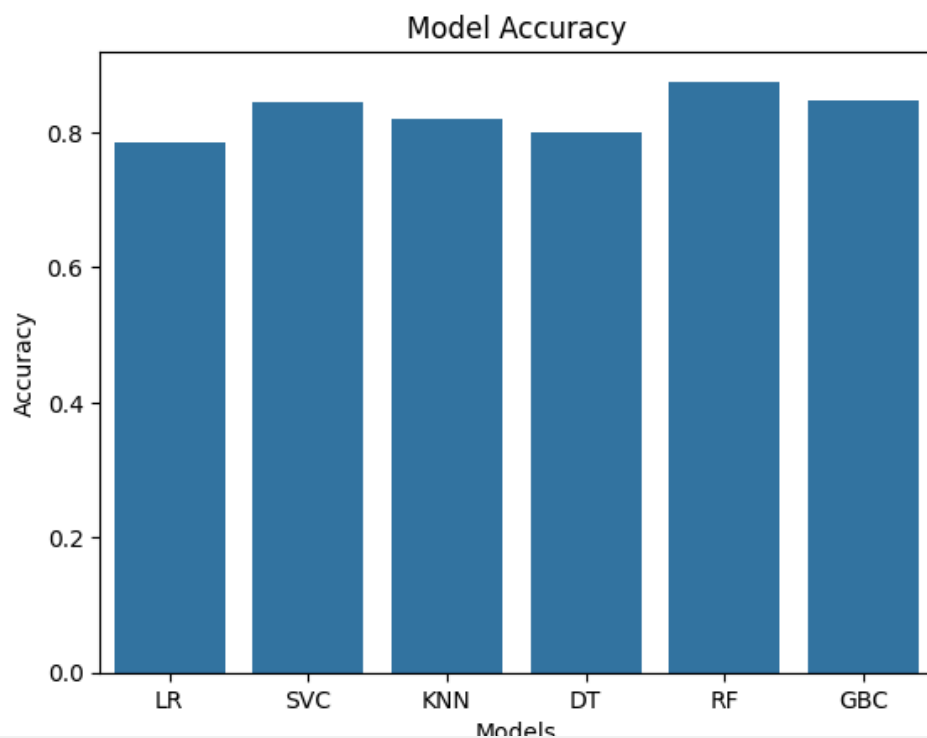
```
plt.xlabel('Models')
```

```
plt.ylabel('Accuracy')
```

```
# Show the plot
```

```
plt.show()
```



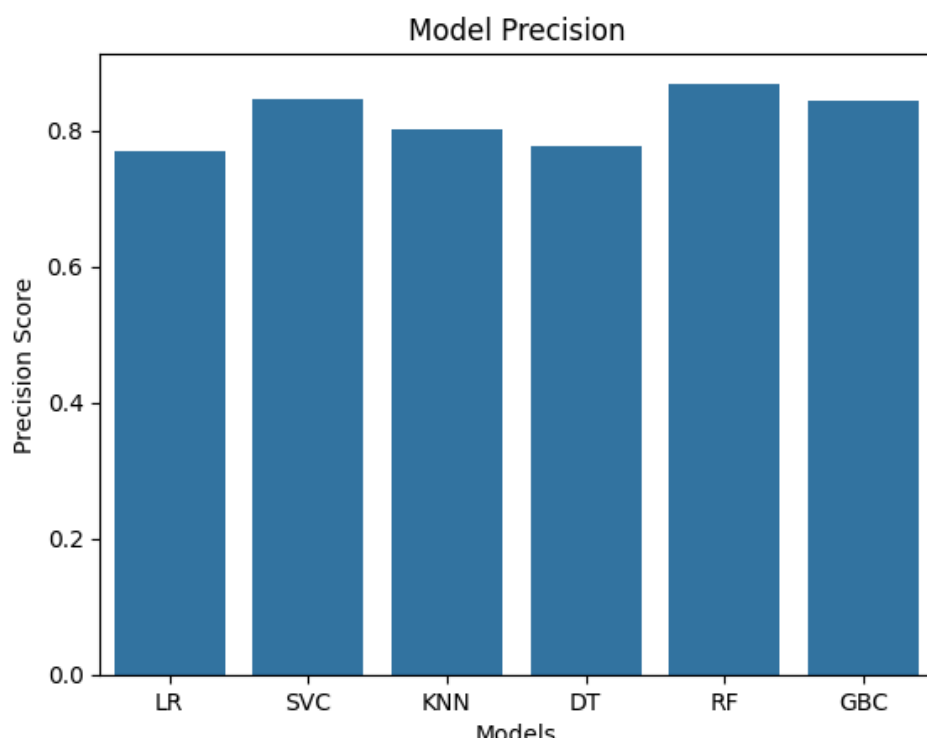


```
import seaborn as sns
import matplotlib.pyplot as plt

# Create a bar plot
sns.barplot(x=final_data['Models'], y=final_data['PRE'])

# Add titles and labels
plt.title('Model Precision')
plt.xlabel('Models')
plt.ylabel('Precision Score')

# Show the plot
plt.show()
```



## 22. Choosing the best ML Algorithm and the saving the model

```
X_res = scaler.fit_transform(X_res)
```

```
rf.fit(X_res, y_res)
```



```
▼ RandomForestClassifier ⓘ ?  
RandomForestClassifier()
```

```
import joblib  
joblib.dump(rf, 'Churn_Predict_Model')
```



```
['Churn_Predict_Model']
```

```
model = joblib.load('Churn_Predict_Model')
```

```
import joblib
joblib.dump(rf, 'Churn_Predict_Model')

↗ [ 'Churn_Predict_Model' ]

model = joblib.load('Churn_Predict_Model')

data.columns

↗ Index(['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'HasCrCard',
        'IsActiveMember', 'EstimatedSalary', 'Exited', 'Geography_Germany',
        'Geography_Spain', 'Gender_Male'],
        dtype='object')

model.predict([[786,32,5,45000,2,1,1,75000,0,1,1]])

↗ array([1])
```