Technological Institute of the Philippines	Quezon City - Computer Engineering
Course Code:	CPE 019
Code Title:	Emerging Technologies in CpE 2 - 2nd Semester
ACTIVITY NO.	**Assignment 7.1 : Classifications and Regression**
Name	Dela Cruz, Irish
Section	CPE32S3
Date Performed:	04/03/2024
Date Submitted:	04/09/2024
Instructor:	Engr. Roman M. Richard

Instructions

1. Choose any dataset applicable to the classification problem, and also, choose any dataset applicable to the regression problem.
2. Explain your datasets and the problem being addressed.
3. For classification, do the following:

- Create a base model
- Evaluate the model with k-fold cross validation
- Improve the accuracy of your model by applying additional hidden layers

4. For regression, do the following:

- Create a base model
- Improve the model by standardizing the dataset
- Show tuning of layers and neurons (see evaluating small and larger networks)

Submit the link to your Google Colab (make sure that it is accessible to me)

REQUIRED RESOURCES

- PC / Laptop with Internet Access
- **Datafiles:** e_comm.csv, Copy of E Commerce Dataset.csv
- Link of dataset: https://data.world/wayvy/customer-churn-dataset-for-a-retail-industry
  https://www.kaggle.com/datasets/ankitverma2010/ecommerce-customer-churn-analysis-and-prediction

Introduction of Dataset

Customer Churn prediction include information about customers, such as their purchase history, demographics, complain, and support interactions using platforms. Each variables (you can see below) provide an insight into various aspects of customer behavior, preferences, and engagement with e-commerce platforms that can used for analysis, prediction, and decision making.

The goal for this dataset is to predict whether a customer is likely churn (cancel their service by stopping the used of plaform) in the future.Early identification of customer at risk of churning allows for targeted rentention strategies. This would be a binary classification problem 1 as churn or 0 as no churn.

EDA (EXPLONATORY DATASET ANALYSIS)

```
!pip install scikeras
```

```
Requirement already satisfied: scikeras in /usr/local/lib/python3.10/dist-packages (0.12.0)
Requirement already satisfied: packaging>=0.21 in /usr/local/lib/python3.10/dist-packages (from scikeras) (24.0)
Requirement already satisfied: scikit-learn>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from scikeras) (1.2.2)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.25.2)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.11.4)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn>=1.0.0->scikeras) (3.4.0)
```

```
!pip install tensorflow
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.15.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=23.5.26 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.5.4)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.9.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes~=0.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: numpy<2.0.0,>=1.23.5 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.25.2)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.20.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (67.7.2)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.10.0)
Requirement already satisfied: wrapt<1.15,>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.14.1)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.36.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.62.1)
Requirement already satisfied: tensorboard<2.16,>=2.15 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.2)
Requirement already satisfied: tensorflow-estimator<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: keras<2.16,>=2.15.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.15.0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (0.43.0)
Requirement already satisfied: google-auth<3,>=1.6.3 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.27.0)
Requirement already satisfied: google-auth-oauthlib<2,>=0.5 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (1.2.0)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (2.31.0)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (0.7.2)
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.16,>=2.15->tensorflow) (3.0.2)
Requirement already satisfied: cachetools<6.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (5.3.3)
Requirement already satisfied: pyasn1-modules>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.4.0)
Requirement already satisfied: rsa<5,>=3.1.4 in /usr/local/lib/python3.10/dist-packages (from google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (4.9)
Requirement already satisfied: requests-oauthlib>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (1.3.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2.21.0->tensorboard<2.16,>=2.15->tensorflow) (2024.2.2)
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.16,>=2.15->tensorflow) (2.1.5)
Requirement already satisfied: pyasn1<0.7.0,>=0.4.6 in /usr/local/lib/python3.10/dist-packages (from pyasn1-modules>=0.2.1->google-auth<3,>=1.6.3->tensorboard<2.16,>=2.15->tensorflow) (0.6.0)
Requirement already satisfied: oauthlib>=3.0.0 in /usr/local/lib/python3.10/dist-packages (from requests-oauthlib>=0.7.0->google-auth-oauthlib<2,>=0.5->tensorboard<2.16,>=2.15->tensorflow) (3.2.2)
```

```
!pip install np_utils
```

```
Requirement already satisfied: np_utils in /usr/local/lib/python3.10/dist-packages (0.6.0)
Requirement already satisfied: numpy>=1.0 in /usr/local/lib/python3.10/dist-packages (from np_utils) (1.25.2)
```

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import RandomOverSampler
from keras.models  import Sequential
from keras.layers import Input, Dense, Flatten, Dropout, BatchNormalization
from keras.optimizers import Adam, SGD, RMSprop

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from keras.models import Sequential
from keras.layers import Dense

from scikeras.wrappers import KerasClassifier
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
```

```
dataset = pd.read_csv("/content/e_comm.csv", na_values="?")
```

```
description = pd.read_csv('/content/Copy of E Commerce Dataset.csv')
description
```

	Data	Variable	Discerption
0	E Comm	CustomerID	Unique customer ID
1	E Comm	Churn	Churn Flag
2	E Comm	Tenure	Tenure of customer in organization
3	E Comm	PreferredLoginDevice	Preferred login device of customer
4	E Comm	CityTier	City tier
5	E Comm	WarehouseToHome	Distance in between warehouse to home of customer
6	E Comm	PreferredPaymentMode	Preferred payment method of customer
7	E Comm	Gender	Gender of customer
8	E Comm	HourSpendOnApp	Number of hours spend on mobile application or...
9	E Comm	NumberOfDeviceRegistered	Total number of deceives is registered on part...
10	E Comm	PreferedOrderCat	Preferred order category of customer in last m...
11	E Comm	SatisfactionScore	Satisfactory score of customer on service
12	E Comm	MaritalStatus	Marital status of customer
13	E Comm	NumberOfAddress	Total number of added added on particular cust...
14	E Comm	Complain	Any complaint has been raised in last month
15	E Comm	OrderAmountHikeFromlastYear	Percentage increases in order from last year
16	E Comm	CouponUsed	Total number of coupon has been used in last m...
17	E Comm	OrderCount	Total number of orders has been places in last...
18	E Comm	DaySinceLastOrder	Day Since last order by customer
19	E Comm	CashbackAmount	Average cashback in last month

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5630 entries, 0 to 5629
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   customerid                   5630 non-null   int64  
 1   churn                        5630 non-null   int64  
 2   tenure                       5366 non-null   float64
 3   preferredlogindevice         5630 non-null   object 
 4   citytier                     5630 non-null   int64  
 5   warehousetohome              5379 non-null   float64
 6   preferredpaymentmode         5630 non-null   object 
 7   gender                       5630 non-null   object 
 8   hourspendonapp               5375 non-null   float64
 9   numberofdeviceregistered     5630 non-null   int64  
 10  preferedordercat             5630 non-null   object 
 11  satisfactionscore            5630 non-null   int64  
 12  maritalstatus                5630 non-null   object 
 13  numberofaddress              5630 non-null   int64  
 14  complain                     5630 non-null   int64  
 15  orderamounthikefromlastyear  5365 non-null   float64
 16  couponused                   5374 non-null   float64
 17  ordercount                   5372 non-null   float64
 18  daysincelastorder            5323 non-null   float64
 19  cashbackamount               5630 non-null   float64
dtypes: float64(8), int64(7), object(5)
memory usage: 879.8+ KB
```

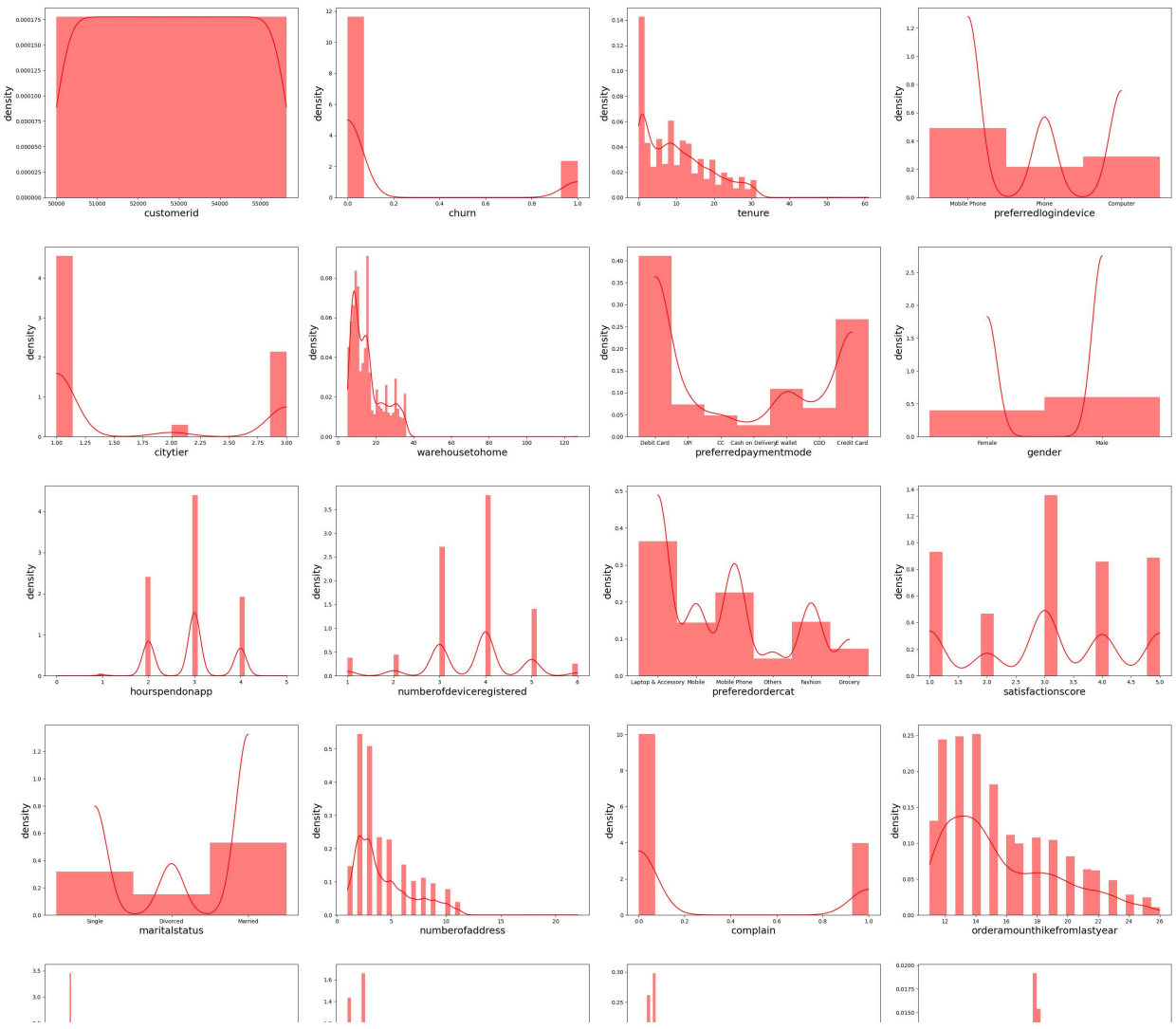
```
dataset.head(20)
```

	customerid	churn	tenure	preferredlogindevice	citytier	warehousetohome	preferredpaymentmode	gender	hourspendonapp	numberofdeviceregistere
0	50001	1	4.000000	1	3	6.0	4	0	3.000000	
1	50002	1	10.189899	2	1	8.0	6	1	3.000000	
2	50003	1	10.189899	2	1	30.0	4	1	2.000000	
3	50004	1	0.000000	2	3	15.0	4	1	2.000000	
4	50005	1	0.000000	2	1	12.0	0	1	2.931535	
5	50006	1	0.000000	0	1	22.0	4	0	3.000000	
6	50007	1	10.189899	2	3	11.0	2	1	2.000000	
7	50008	1	10.189899	2	1	6.0	0	1	3.000000	
8	50009	1	13.000000	2	3	9.0	5	1	2.931535	
9	50010	1	10.189899	2	1	31.0	4	1	2.000000	
10	50011	1	4.000000	1	1	18.0	2	0	2.000000	
11	50012	1	11.000000	1	1	6.0	4	1	3.000000	
12	50013	1	0.000000	2	1	11.0	1	1	2.000000	
13	50014	1	0.000000	2	1	15.0	0	1	3.000000	
14	50015	1	9.000000	1	3	15.0	3	1	3.000000	
15	50016	1	10.189899	2	2	12.0	6	1	3.000000	
16	50017	1	0.000000	0	1	12.0	4	0	2.931535	
17	50018	1	0.000000	1	3	11.0	5	1	2.000000	
18	50019	1	0.000000	0	1	13.0	4	1	3.000000	
19	50020	1	19.000000	1	1	20.0	4	0	3.000000	

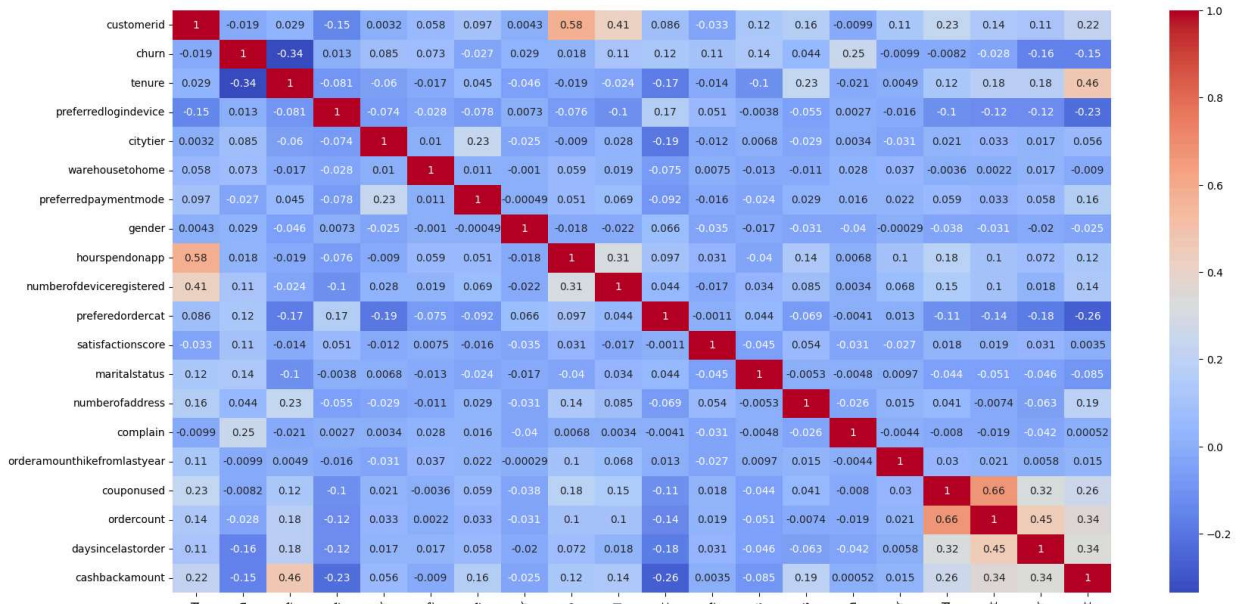
```
dataset.tail(20)
```

	customerid	churn	tenure	preferredlogindevice	citytier	warehousetohome	preferredpaymentmode	gender	hourspendonapp	numberofdeviceregistere
5610	55611	0	9.0	1	2	33.0	4	0	3.0	
5611	55612	0	12.0	1	3	17.0	4	0	4.0	
5612	55613	1	14.0	0	3	8.0	4	1	4.0	
5613	55614	0	10.0	1	3	18.0	5	0	4.0	
5614	55615	0	19.0	1	3	27.0	4	0	3.0	
5615	55616	0	14.0	1	1	9.0	4	1	1.0	
5616	55617	0	9.0	0	1	25.0	4	1	4.0	
5617	55618	0	14.0	2	1	9.0	3	0	4.0	
5618	55619	0	9.0	1	1	8.0	4	0	4.0	
5619	55620	0	3.0	1	1	20.0	6	1	3.0	
5620	55621	0	3.0	1	1	35.0	3	0	4.0	
5621	55622	1	14.0	1	3	35.0	5	1	3.0	
5622	55623	0	13.0	1	3	31.0	5	0	3.0	
5623	55624	0	5.0	0	1	12.0	3	1	4.0	
5624	55625	0	1.0	1	3	12.0	6	0	2.0	
5625	55626	0	10.0	0	1	30.0	3	1	3.0	
5626	55627	0	13.0	1	1	13.0	3	1	3.0	
5627	55628	0	1.0	1	1	11.0	4	1	3.0	
5628	55629	0	23.0	0	3	9.0	3	1	4.0	
5629	55630	0	8.0	1	1	15.0	3	1	3.0	

```
num_cols = min(len(dataset.columns), 4)
num_rows = (len(dataset.columns) + num_cols - 1) // num_cols

fig, ax = plt.subplots(ncols=num_cols, nrows= num_rows, figsize=(30, 30))
index = 0
ax = ax.flatten()

for col, value in dataset.items():
    col_dist = sns.histplot(value, ax=ax[index], color='red', kde=True, stat="density", linewidth=0)
    col_dist.set_xlabel(col, fontsize=18)
    col_dist.set_ylabel('density', fontsize=18)
    index += 1

plt.tight_layout(pad=0.5, w_pad=0.7, h_pad=5.0)
```




```
plt.figure(figsize=(20, 10))
sns.heatmap(dataset.corr(), annot=True, cmap='coolwarm')
plt.show()
```




```
dataset['churn'].value_counts()
```

```
churn
0    4682
1     948
Name: count, dtype: int64
```

```
churn_by_prefered_ordercat = pd.crosstab(dataset['preferedordercat'], dataset['churn'])
print(churn_by_prefered_ordercat)
```

```
churn                  0    1
preferedordercat
Fashion              698  128
Grocery              390   20
Laptop & Accessory  1840  210
Mobile               589  220
Mobile Phone         921  350
Others               244   20
```

```
churn_by_tenure = pd.crosstab(dataset['tenure'], dataset['churn'])
print(churn_by_tenure)
```

```
churn     0    1
tenure
0.0     236  272
1.0     341  349
2.0     153   14
3.0     177   18
```

```
4.0     183   20
5.0     188   16
6.0     175    8
7.0     205   16
8.0     247   16
9.0     235   12
10.0    199   14
11.0    184   10
12.0    175    7
13.0    168   13
14.0    162   14
15.0    149   10
16.0    139   10
17.0    102    4
18.0    117    6
19.0    128   12
20.0     93   16
21.0     74   10
22.0     76    0
23.0     89    0
24.0     75    0
25.0     59    0
26.0     60    0
27.0     66    0
28.0     70    0
29.0     55    0
30.0     66    0
31.0     49    0
50.0      1    0
51.0      1    0
60.0      1    0
61.0      1    0
```

Data Pre-Processing

```
dataset.isnull().sum()
```

```
customerid                      0
churn                           0
tenure                        264
preferredlogindevice            0
citytier                        0
warehousetohome               251
preferredpaymentmode            0
gender                          0
hourspendonapp                255
numberofdeviceregistered        0
preferedordercat                0
satisfactionscore               0
maritalstatus                   0
numberofaddress                 0
complain                        0
orderamounthikefromlastyear    265
couponused                     256
ordercount                     258
daysincelastorder              307
cashbackamount                   0
dtype: int64
```

```
missing_values = dataset.isnull().sum().sum()
rows_with_missing_values = dataset.isnull().any(axis=1).sum()

print(f'The number of missing values: {missing_values}')
print(f'The number of rows with missing values: {rows_with_missing_values}')
```

```
The number of missing values: 1856
The number of rows with missing values: 1856
```

```
dataset["tenure"].fillna(dataset["tenure"].mean(), inplace=True)
dataset["warehousetohome"].fillna(dataset["warehousetohome"].mean(), inplace=True)
dataset["hourspendonapp"].fillna(dataset["hourspendonapp"].mean(), inplace=True)
dataset["orderamounthikefromlastyear"].fillna(dataset["orderamounthikefromlastyear"].mean(), inplace=True)
dataset["couponused"].fillna(dataset["couponused"].mean(), inplace=True)
dataset["ordercount"].fillna(dataset["ordercount"].mean(), inplace=True)
dataset["daysincelastorder"].fillna(dataset["daysincelastorder"].mean(), inplace=True)
```

```
dataset.isnull().sum()
```

```
customerid                    0
churn                         0
tenure                        0
preferredlogindevice          0
citytier                      0
warehousetohome               0
preferredpaymentmode          0
gender                        0
hourspendonapp                0
numberofdeviceregistered      0
preferedordercat              0
satisfactionscore             0
maritalstatus                 0
numberofaddress               0
complain                      0
orderamounthikefromlastyear   0
couponused                    0
ordercount                    0
daysincelastorder             0
cashbackamount                0
dtype: int64
```

```
dataset.describe()
```

	customerid	churn	tenure	citytier	warehousetohome	hourspendonapp	numberofdeviceregistered	satisfactionscore	numberofaddress
count	5630.000000	5630.000000	5630.000000	5630.000000	5630.000000	5630.000000	5630.000000	5630.000000	5630.000000
mean	52815.500000	0.168384	10.189899	1.654707	15.639896	2.931535	3.688988	3.066785	4.214032
std	1625.385339	0.374240	8.354164	0.915389	8.339095	0.705384	1.023999	1.380194	2.583586
min	50001.000000	0.000000	0.000000	1.000000	5.000000	0.000000	1.000000	1.000000	1.000000
25%	51408.250000	0.000000	3.000000	1.000000	9.000000	2.000000	3.000000	2.000000	2.000000
50%	52815.500000	0.000000	9.000000	1.000000	14.000000	3.000000	4.000000	3.000000	3.000000
75%	54222.750000	0.000000	15.000000	3.000000	20.000000	3.000000	4.000000	4.000000	6.000000
max	55630.000000	1.000000	61.000000	3.000000	127.000000	5.000000	6.000000	5.000000	22.000000

```
dataset.shape
```

```
(5630, 20)
```

```
Zero = dataset[dataset.churn == 0] # no purchase
One = dataset[dataset.churn == 1]  # purchase
```

```
ZeroDS = Zero.sample(len(One), replace = True, random_state=100)
OneDB = pd.concat([ZeroDS, One])
```

```
count = OneDB['churn'].value_counts()
print(count)
```

```
churn
0    948
1    948
Name: count, dtype: int64
```

```
numerical_features = dataset.select_dtypes(include=['float64', 'int64'])
categorical_features = dataset.select_dtypes(include=['object'])

print("Numerical Features:")
print(numerical_features.columns)

print("\nCategorical Features:")
print(categorical_features.columns)
```

```
Numerical Features:
Index(['customerid', 'churn', 'tenure', 'citytier', 'warehousetohome',
       'hourspendonapp', 'numberofdeviceregistered', 'satisfactionscore',
       'numberofaddress', 'complain', 'orderamounthikefromlastyear',
       'couponused', 'ordercount', 'daysincelastorder', 'cashbackamount'],
      dtype='object')
```

```
Categorical Features:
Index(['preferredlogindevice', 'preferredpaymentmode', 'gender',
       'preferedordercat', 'maritalstatus'],
      dtype='object')
```

Handling Categorical Variable

```
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
dataset['preferredlogindevice'] = label_encoder.fit_transform(dataset['preferredlogindevice'])
dataset['preferredpaymentmode'] = label_encoder.fit_transform(dataset['preferredpaymentmode'])
dataset['gender'] = label_encoder.fit_transform(dataset['gender'])
dataset['preferedordercat'] = label_encoder.fit_transform(dataset['preferedordercat'])
dataset['maritalstatus'] = label_encoder.fit_transform(dataset['maritalstatus'])
dataset.head(20)
```

	customerid	churn	tenure	preferredlogindevice	citytier	warehousetohome	preferredpaymentmode	gender	hourspendonapp	numberofdeviceregistere
0	50001	1	4.000000	1	3	6.0	4	0	3.000000	
1	50002	1	10.189899	2	1	8.0	6	1	3.000000	
2	50003	1	10.189899	2	1	30.0	4	1	2.000000	
3	50004	1	0.000000	2	3	15.0	4	1	2.000000	
4	50005	1	0.000000	2	1	12.0	0	1	2.931535	
5	50006	1	0.000000	0	1	22.0	4	0	3.000000	
6	50007	1	10.189899	2	3	11.0	2	1	2.000000	
7	50008	1	10.189899	2	1	6.0	0	1	3.000000	
8	50009	1	13.000000	2	3	9.0	5	1	2.931535	
9	50010	1	10.189899	2	1	31.0	4	1	2.000000	
10	50011	1	4.000000	1	1	18.0	2	0	2.000000	
11	50012	1	11.000000	1	1	6.0	4	1	3.000000	
12	50013	1	0.000000	2	1	11.0	1	1	2.000000	
13	50014	1	0.000000	2	1	15.0	0	1	3.000000	
14	50015	1	9.000000	1	3	15.0	3	1	3.000000	
15	50016	1	10.189899	2	2	12.0	6	1	3.000000	
16	50017	1	0.000000	0	1	12.0	4	0	2.931535	
17	50018	1	0.000000	1	3	11.0	5	1	2.000000	
18	50019	1	0.000000	0	1	13.0	4	1	3.000000	
19	50020	1	19.000000	1	1	20.0	4	0	3.000000	

Remarks

I used label encoder here, since neural networks is crucial for enabling models to effectively learn from non-numerical data, improve performance, and avoid bias. So proper encoding and representation of categorical variables allow neural networks to leverage the full potential of the available data and make more accurate predictions

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5630 entries, 0 to 5629
Data columns (total 20 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   customerid                   5630 non-null   int64  
 1   churn                        5630 non-null   int64  
 2   tenure                       5630 non-null   float64
 3   preferredlogindevice         5630 non-null   int64  
 4   citytier                     5630 non-null   int64  
 5   warehousetohome              5630 non-null   float64
 6   preferredpaymentmode         5630 non-null   int64  
 7   gender                       5630 non-null   int64  
 8   hourspendonapp               5630 non-null   float64
 9   numberofdeviceregistered     5630 non-null   int64  
 10  preferedordercat             5630 non-null   int64  
 11  satisfactionscore            5630 non-null   int64  
 12  maritalstatus                5630 non-null   int64  
 13  numberofaddress              5630 non-null   int64  
 14  complain                     5630 non-null   int64  
 15  orderamounthikefromlastyear  5630 non-null   float64
 16  couponused                   5630 non-null   float64
 17  ordercount                   5630 non-null   float64
 18  daysincelastorder            5630 non-null   float64
 19  cashbackamount               5630 non-null   float64
dtypes: float64(8), int64(12)
memory usage: 879.8 KB
```

```
dataset.columns
```

```
Index(['customerid', 'churn', 'tenure', 'preferredlogindevice', 'citytier',
       'warehousetohome', 'preferredpaymentmode', 'gender', 'hourspendonapp',
       'numberofdeviceregistered', 'preferedordercat', 'satisfactionscore',
       'maritalstatus', 'numberofaddress', 'complain',
       'orderamounthikefromlastyear', 'couponused', 'ordercount',
       'daysincelastorder', 'cashbackamount'],
      dtype='object')
```

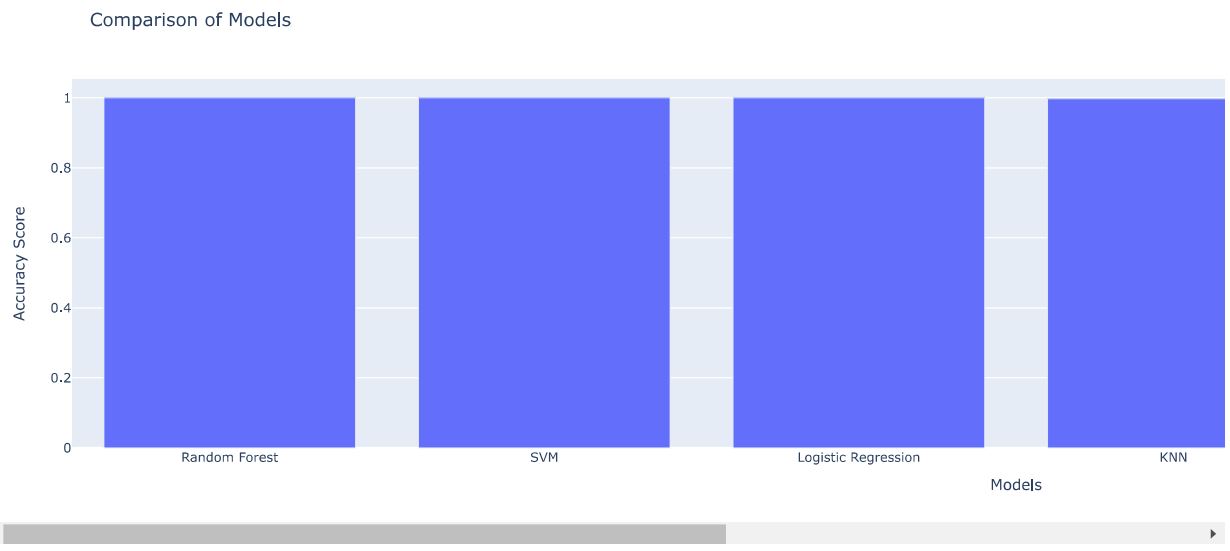
```
X = dataset.iloc[:, :-1].values
y = dataset["churn"].values
```

```
X = dataset.drop(columns=['churn'])
y = dataset['churn']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=2)


from tensorflow.keras.utils import to_categorical
y_train_encoded = to_categorical(y_train, num_classes=19)


normalizer = StandardScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)


np.mean(y), np.mean(1-y)
```

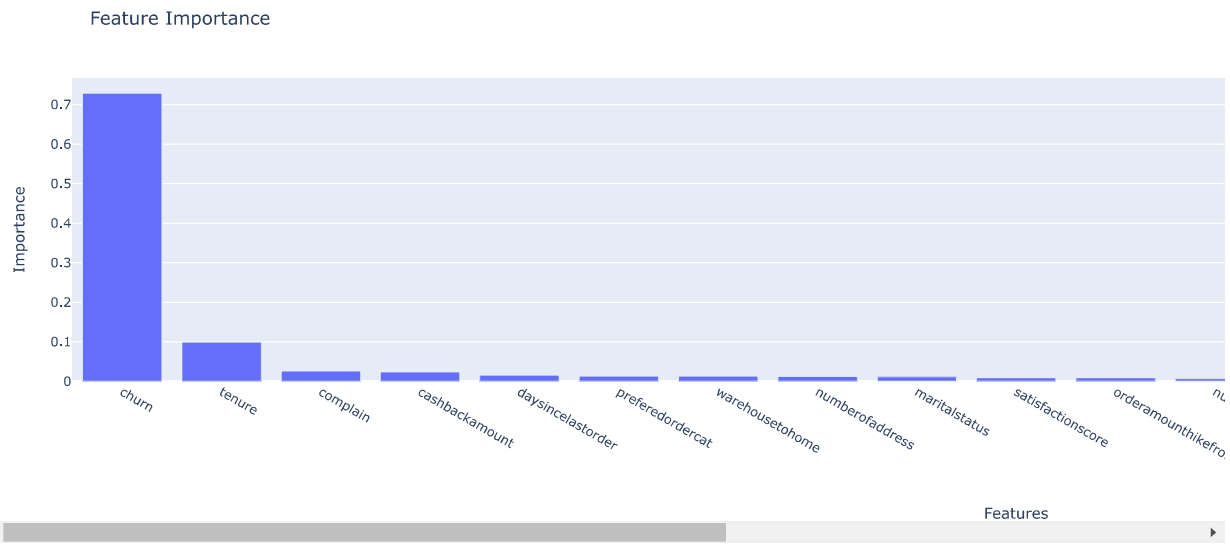
```
(0.16838365896980462, 0.8316163410301953)
```

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', RandomForestClassifier())
])
```

```
param_grid = [
    {
        'clf': [RandomForestClassifier()],
        'clf__n_estimators': [100, 200, 300,400],
        'clf__max_depth': [None, 5, 10,15],
    },
    {
        'clf': [SVC()],
        'clf__kernel': ['linear', 'rbf'],
        'clf__C': [0.01,0.1, 1, 10],
    },
    {
        'clf': [LogisticRegression()],
        'clf__solver': ['liblinear', 'lbfgs'],
        'clf__C': [0.01,0.1, 1, 10],
    },
    {
        'clf': [KNeighborsClassifier()],
        'clf__n_neighbors': [3, 5, 7,9],
    },
    {
        'clf': [GradientBoostingClassifier()],
        'clf__n_estimators': [100, 200, 300,400],
        'clf__learning_rate': [0.01, 0.1, 1],
    },
    {
        'clf': [DecisionTreeClassifier()],
        'clf__max_depth': [None, 5, 10,15],
    }
]


grid_search = GridSearchCV(pipeline, param_grid, cv=5)
grid_search.fit(X_train, y_train)

best_model = grid_search.best_estimator_
models = [
    ('Random Forest', RandomForestClassifier()),
    ('SVM', SVC()),
    ('Logistic Regression', LogisticRegression()),
    ('KNN', KNeighborsClassifier()),
    ('Gradient Boosting', GradientBoostingClassifier()),
    ('Decision Tree', DecisionTreeClassifier())
]

accuracy_scores = []
for name, model in models:
    pipeline = Pipeline([
        ('scaler', StandardScaler()),
          ('clf', model)
    ])
    pipeline.fit(X_train, y_train)
    y_pred = pipeline.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_scores.append(accuracy)


import plotly.graph_objects as go

fig = go.Figure(data=go.Bar(x=[name for name, _ in models], y=accuracy_scores))
fig.update_layout(title='Comparison of Models',
                  xaxis=dict(title='Models'),
                  yaxis=dict(title='Accuracy Score'))
fig.show()
```




```
importance = best_model.named_steps['clf'].feature_importances_
feature_names = X.columns

sorted_indices = np.argsort(importance)[::-1]
sorted_importance = importance[sorted_indices]
sorted_features = feature_names[sorted_indices]

fig = go.Figure(data=go.Bar(x=sorted_features, y=sorted_importance))
fig.update_layout(title='Feature Importance',
                  xaxis=dict(title='Features'),
                  yaxis=dict(title='Importance'))
fig.show()
```



Remarks for Feature Importance

As you can see above the Churn got a higher score among the rest which most important feature for predicting customer satisfaction that lead whether the customer will stay or leave. The tenure and complain also may be a factor to influence the customer satisfaction that's why its very important to make your customer satisfied with the platform they used because they're more likely be satisfied again in the future.

Model for Classification

```
print("Shapes of X_train, X_test, y_train, y_test:", [arr.shape for arr in (X_train, X_test, y_train, y_test)])
```

```
Shapes of X_train, X_test, y_train, y_test: [(4504, 19), (1126, 19), (4504,), (1126,)]
```

```
def baseline_model():
```

```
    model = Sequential()
    model.add(Dense(80, input_dim=19, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(80, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1,  activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Evaluation of Model with K-Fold Cross Validation

```
estimator = KerasClassifier(model=baseline_model, epochs=200, batch_size=200, verbose=0)
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator, X, y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Baseline: 83.16% (1.61%)
```

Remarks for Classification

The mean accuracy of the baseline model, indicates that the model correctIt preducts the target variable as 83.16% of the instances in the dataset. The STD is 1.61% of accuracu accross different iterations or 10 k-fold, which provides information about the variablitiy in model's performance across different subsets of data or training /testing sets

Improvement of Model

```
def baseline_model():
    model = Sequential()
    model.add(Dense(50, input_dim=19, activation='relu'))
    model.add(Dropout(0.7))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(60, activation='relu'))
    model.add(Dense(1,  activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

```
estimator = KerasClassifier(model=baseline_model, epochs=200, batch_size=200, verbose=0)
kfold = KFold(n_splits=10, shuffle=True)
results = cross_val_score(estimator, X, y, cv=kfold)
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Baseline: 83.16% (1.72%)
```

Remarks for Improvement Classification

The mean accuracy of the baseline model, indicates that the model correctIt preducts the target variable as 83.16% of the instances in the dataset. The STD is 1.48% of accuracu accross different iterations or 10 k-fold, which provides information about the variablitiy in model's performance across different subsets of data or training /testing sets

Models for Regression

Baseline

```
def baseline_model():
  model = Sequential()
  model.add(Dense(20, input_shape=(19,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
  model.add(Dense(20, kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model
```

```
from sklearn.model_selection import KFold, cross_val_score
from scikeras.wrappers import KerasRegressor

estimator = KerasRegressor(model=baseline_model, epochs=100, batch_size=5, verbose=0)
kfold = KFold(n_splits=10)
results = cross_val_score(estimator, X, y, cv=kfold, scoring='neg_mean_squared_error')
print("Baseline: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Baseline: -0.14 (0.01) MSE
```

Remarks for Baseline Regression

When executing this code, it provides an estimate of -0.14 as the model's performance on unseen data. This result represents the mean squared error, incorporating both the average and standard deviation across all 10 folds of the cross-validation evaluation.

Standardized

```
def standardized_model():
  model = Sequential()
  model.add(Dense(30, input_shape=(19,), kernel_initializer='normal', activation='relu'))
  model.add(Dense(20,  kernel_initializer='normal', activation='relu'))
  model.add(Dense(10,  kernel_initializer='normal', activation='relu'))
  model.add(Dense(1, kernel_initializer='normal'))
  model.compile(loss='mean_squared_error', optimizer='adam')
  return model
```

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=standardized_model, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, y, cv=kfold, scoring='neg_mean_squared_error')
print("Standardized: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Standardized: -0.04 (0.01) MSE
```

Remarks for Standardized Regression

Upon running this model, there was a notable improvement in performance, with the mean squared error decreasing from 14 to 4. The negative mean squared error, though initially counterintuitive, actually denotes positive performance, where lower values signify better model performance.

Larger

```
def larger_model():
    model = Sequential()
    model.add(Dense(50, input_shape=(19,), kernel_initializer='normal', activation='relu'))
    model.add(Dense(3, kernel_initializer='normal', activation='relu'))
    model.add(Dense(1, kernel_initializer='normal'))
    model.compile(loss='mean_squared_error', optimizer='adam')
    return model
```

```
estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=larger_model, epochs=50, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, y, cv=kfold, scoring='neg_mean_squared_error')
print("Larger: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Larger: -0.05 (0.01) MSE
```

Remarks for Standardized Regression

Executing the provided code yields an enhanced performance compared to the baseline model, evident in a reduction of the error by 10 k-folds. Since it split into 10 subnets and will be trained and evaluate 10 times with a result of -0.05 that helps me providing a more robust estimate of model's performance by averaging the results over multiple iterations

Wider

```
def wider_model():
 model = Sequential()
 model.add(Dense(30, input_shape=(19,), kernel_initializer='normal', activation='relu'))
 model.add(Dense(50, kernel_initializer='normal', activation='relu'))
 model.add(Dense(30, kernel_initializer='normal', activation='relu'))
 model.add(Dense(1, kernel_initializer='normal'))
 model.compile(loss='mean_squared_error', optimizer='adam')
 return model


estimators = []
estimators.append(('standardize', StandardScaler()))
estimators.append(('mlp', KerasRegressor(model=wider_model, epochs=100, batch_size=5, verbose=0)))
pipeline = Pipeline(estimators)
kfold = KFold(n_splits=10)
results = cross_val_score(pipeline, X, y, cv=kfold, scoring='neg_mean_squared_error')
print("Wider: %.2f (%.2f) MSE" % (results.mean(), results.std()))
```

```
Wider: -0.02 (0.01) MSE
```

Remarks for Wider Regression

The development of a model reveals a notable decrease in error, down to 2 k-folds. This outcome is commendable, considering the unexpected performance of a wider network compared to a deeper one in this scenario. These results underscore the significance of empirical testing in the process of neural network model development.