Featured examples

- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

Importing The Dependancies

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

```
#loading the dataset to a Panda's Data Frame
credit_card = pd.read_csv('/content/creditcard.csv')
```

```
#first 5 rows of the dataset
credit_card.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.0986
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.0851
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.2476
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.3774
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.2705

5 rows × 31 columns

```
credit_card.tail()
```

	Time	V1	V2	V3	V4	V5	V6	
284802	172786.0	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-2.606837	-4.918
284803	172787.0	-0.732789	-0.055080	2.035030	-0.738589	0.868229	1.058415	0.024
284804	172788.0	1.919565	-0.301254	-3.249640	-0.557828	2.630515	3.031260	-0.296
284805	172788.0	-0.240440	0.530483	0.702510	0.689799	-0.377961	0.623708	-0.686
284806	172792.0	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	-0.649617	1.577

5 rows × 31 columns

```
#dataset_informations
credit_card.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
```

```
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```
#check the number of missing values in each coloumn
credit_card.isnull().sum()
```

```
Time      0
V1        0
V2        0
V3        0
V4        0
V5        0
V6        0
V7        0
V8        0
V9        0
V10       0
V11       0
V12       0
V13       0
V14       0
V15       0
V16       0
V17       0
V18       0
V19       0
V20       0
V21       0
V22       0
V23       0
V24       0
V25       0
V26       0
V27       0
V28       0
Amount    0
Class     0
dtype: int64
```

```
#distribution of legit transactions and fraudalent transactions
credit_card['Class'].value_counts()
```

```
Class
0    284315
1       492
Name: count, dtype: int64
```

This Dataset is highly Unbalanced

0--->Normal Transaction 1--->Fraudalent Transaction

```
#seperating the data for analysis
legit=credit_card[credit_card.Class==0]
fraud=credit_card[credit_card.Class==1]
```

```
print(legit.shape)
print(fraud.shape)
```

```
(284315, 31)
(492, 31)
```

```
#statistical measures of the data
legit.Amount.describe()
```

```
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

```
fraud.Amount.describe()
```

```
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
```

```
#COMPARE THE VALUES FOR BOTH THE TRANSACTIONS
credit_card.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	
0	94838.202258	0.008258	-0.006271	0.012171	-0.007860	0.005453	0.002419	0.0(
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.5(

2 rows × 30 columns

Under-Sampling

Build a sample dataset containing similar distribution of normal transactions and fraudelent transactions. Number of Fraudalent Transactions---> 492

```
legit_sample=legit.sample(n=492)
```

Concatenating two DataFrames

```
new_dataset=pd.concat([legit_sample,fraud],axis=0)
```

```
new_dataset.head()
```

	Time	V1	V2	V3	V4	V5	V6	\
54392	46428.0	-0.514821	0.521004	1.264503	-0.632441	0.862013	-1.025945	1.1321(
195042	130872.0	1.690675	-1.510145	-1.600421	-0.246420	-0.760748	-0.557438	-0.1978ł
228100	145359.0	1.941254	0.116085	-1.728416	1.231754	0.644916	-0.692562	0.6591:
46044	42614.0	-0.740093	1.112070	1.356729	0.906086	-0.061893	0.131743	0.0939:
98413	66652.0	-0.771862	1.243432	1.138354	0.020260	0.002454	-1.035237	0.6679(

5 rows × 31 columns

```
new_dataset.tail()
```

	Time	V1	V2	V3	V4	V5	V6	V7
279863	169142.0	-1.927883	1.125653	-4.518331	1.749293	-1.566487	-2.010494	-0.88285
280143	169347.0	1.378559	1.289381	-5.004247	1.411850	0.442581	-1.326536	-1.41317
280149	169351.0	-0.676143	1.126366	-2.213700	0.468308	-1.120541	-0.003346	-2.23473
281144	169966.0	-3.113832	0.585864	-5.399730	1.817092	-0.840618	-2.943548	-2.20800
281674	170348.0	1.991976	0.158476	-2.583441	0.408670	1.151147	-0.096695	0.22305

5 rows × 31 columns

```
new_dataset['Class'].value_counts()
```

```
Class
0    492
1    492
Name: count, dtype: int64
```

```
new_dataset.groupby('Class').mean()
```

Class	Time	V1	V2	V3	V4	V5	V6	
0	91792.532520	0.173670	-0.040220	0.083540	0.016815	-0.067310	0.024421	-0.04
1	80746.806911	-4.771948	3.623778	-7.033281	4.542029	-3.151225	-1.397737	-5.56

2 rows × 30 columns

Splitting the data into Features & Targets

```
X=new_dataset.drop(columns='Class',axis=1)
Y=new_dataset['Class']
```

```
print(X)
```

```
            Time        V1        V2        V3        V4        V5        V6  \
54392    46428.0 -0.514821  0.521004  1.264503 -0.632441  0.862013 -1.025945
195042  130872.0  1.690675 -1.510145 -1.600421 -0.246420 -0.760748 -0.557438
228100  145359.0  1.941254  0.116085 -1.728416  1.231754  0.644916 -0.692562
46044    42614.0 -0.740093  1.112070  1.356729  0.906086 -0.061893  0.131743
98413    66652.0 -0.771862  1.243432  1.138354  0.020260  0.002454 -1.035237
...          ...       ...       ...       ...       ...       ...       ...
279863  169142.0 -1.927883  1.125653 -4.518331  1.749293 -1.566487 -2.010494
280143  169347.0  1.378559  1.289381 -5.004247  1.411850  0.442581 -1.326536
280149  169351.0 -0.676143  1.126366 -2.213700  0.468308 -1.120541 -0.003346
281144  169966.0 -3.113832  0.585864 -5.399730  1.817092 -0.840618 -2.943548
281674  170348.0  1.991976  0.158476 -2.583441  0.408670  1.151147 -0.096695

              V7        V8        V9  ...       V20       V21       V22  \
54392   1.132102 -0.456591 -0.653587  ...  0.277210 -0.383096 -1.112879
195042 -0.197887 -0.146472 -0.290996  ... -0.274700  0.105502  0.351378
228100  0.659122 -0.336300 -0.167197  ... -0.139273  0.108472  0.452649
46044   0.093921  0.559996 -0.706923  ... -0.040523 -0.069077 -0.172177
98413   0.667960 -0.091593 -0.096800  ...  0.167565 -0.296246 -0.715803
...          ...       ...       ...  ...       ...       ...       ...
279863 -0.882850  0.697211 -2.064945  ...  1.252967  0.778584 -0.319189
280143 -1.413170  0.248525 -1.127396  ...  0.226138  0.370612  0.028234
280149 -2.234739  1.210158 -0.652250  ...  0.247968  0.751826  0.834108
281144 -2.208002  1.058733 -1.632333  ...  0.306271  0.583276 -0.269209
281674  0.223050 -0.068384  0.577829  ... -0.017652 -0.164350 -0.295135

             V23       V24       V25       V26       V27       V28  Amount
54392   0.042970 -0.089883 -0.339500  0.574005 -0.181567 -0.115154   29.95
195042 -0.201214 -0.421157  0.073151 -0.031675 -0.043053 -0.031120  237.00
228100 -0.105558 -0.305245  0.499747 -0.499299 -0.024020 -0.066169   48.20
46044  -0.128324 -0.037479 -0.124108 -0.483832  0.045495  0.054597    2.68
98413   0.095061  0.323654 -0.102961  0.069698  0.239995  0.140031    3.58
...          ...       ...       ...       ...       ...       ...     ...
279863  0.639419 -0.294885  0.537503  0.788395  0.292680  0.147968  390.00
280143 -0.145640 -0.081049  0.521875  0.739467  0.389152  0.186637    0.76
280149  0.190944  0.032070 -0.739695  0.471111  0.385107  0.194361   77.89
281144 -0.456108 -0.183659 -0.328168  0.606116  0.884876 -0.253700  245.00
281674 -0.072173 -0.450261  0.313267 -0.289617  0.002988 -0.015309   42.53

[984 rows x 30 columns]
```

```
print(Y)
```

```
54392     0
195042    0
228100    0
46044     0
98413     0
         ..
279863    1
280143    1
280149    1
281144    1
281674    1
Name: Class, Length: 984, dtype: int64
```

Split the data into Training data & testing data

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,stratify=Y,random_state=2)
```

```
print(X.shape,X_train.shape,X_test.shape)
```

```
(984, 30) (787, 30) (197, 30)
```

Model training

Logistic Regression

```
#training the Logistic Regression Model with Training data
model=LogisticRegression()
```

```
#training the Logistic Regression Model with Training Data
model.fit(X_train,Y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

Evaluation of the Model Accuracy Score

```
#accuracy on training data
X_train_prediction=model.predict(X_train)
training_data_accuracy=accuracy_score(X_train_prediction,Y_train)
```

```
print('Accuracy on Training data : ',training_data_accuracy)
```

```
Accuracy on Training data :  0.9504447268106735
```

```
#accuracy on test data
X_test_prediction=model.predict(X_test)
test_data_accuracy=accuracy_score(X_test_prediction,Y_test)
```

```
print('Accuracy score on Test Data:',test_data_accuracy)
```

```
Accuracy score on Test Data: 0.9390862944162437
```