```
! pip install kaggle
! mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json

! kaggle competitions download loan-default-prediction
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simp
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from k
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from rec
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (fro
mkdir: cannot create directory ‘/root/.kaggle’: File exists
loan-default-prediction.zip: Skipping, found more recently modified local copy (use --force to
```

```
! unzip /content/loan-default-prediction.zip
```

```
Archive:  /content/loan-default-prediction.zip
replace sampleSubmission.csv? [y]es, [n]o, [A]ll, [N]one, [r]ename: N
```

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split
```

```
data = pd.read_csv('/content/train_v2.csv.zip')
data.head()
```

```
/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py:3326: DtypeWarning: Co
  exec(code_obj, self.user_global_ns, self.user_ns)
```

	id	f1	f2	f3	f4	f5	f6	f7	f8	f9	...	f770	f771	f772	f773
0	1	126	10	0.686842	1100	3	13699	7201.0	4949.0	126.75	...	5	2.14	-1.54	1.18
1	2	121	10	0.782776	1100	3	84645	240.0	1625.0	123.52	...	6	0.54	-0.24	0.13
2	3	126	10	0.500080	1100	3	83607	1800.0	1527.0	127.76	...	13	2.89	-1.73	1.04
3	4	134	10	0.439874	1100	3	82642	7542.0	1730.0	132.94	...	4	1.29	-0.89	0.66
4	5	109	9	0.502749	2900	4	79124	89.0	491.0	122.72	...	26	6.11	-3.82	2.51

5 rows × 771 columns

```
data.shape
```

```
(105471, 771)
```

```
len(data.loss.unique())
```

```
89
```

```
# Remove all classes which have only 3 or less observations
for i in data.loss.unique():
  if data.loc[data['loss'] == i].shape[0] <= 3:
    index_names = data[data['loss'] == i ].index
    data.drop(index_names, inplace = True)
```

```
len(data.loss.unique())
```

```
63
```

```
data.shape
```

```
(105430, 771)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 105430 entries, 0 to 105470
Columns: 771 entries, id to loss
dtypes: float64(653), int64(99), object(19)
memory usage: 621.0+ MB
```

```
# Select columns with 'object' datatype
data.select_dtypes(include=['object']).head()
```

	f137	f138	f206	f207	
0	8090000000000000	754485076006959972352	3200000000000	38600000000000000	7900000000
1	2250000000000	15300000000000000	392000000000	1690000000000000	92300000
2	186000000000000	6910365323840000000	23700000000000	389000000000000000	10300000
3	44500000000000000	11225194901267999096832	16098514954	35000000000000	22200000
4	52152926246	108000000000000	442000000000	1870000000000000	3630000

```
# Drop all coulmns with 'object' datatype
for i in data.select_dtypes(include=['object']).columns:
    data.drop(labels=i, axis=1, inplace=True)
```

```
data.shape
```

```
(105430, 752)
```

```
data.describe()
```

	id	f1	f2	f3	f4	f5
count	105430.000000	105430.000000	105430.000000	105430.000000	105430.000000	105430.000000
mean	52737.898075	134.603102	8.246846	0.499089	2678.440672	7.354975
std	30447.038488	14.725194	1.691596	0.288746	1400.917228	5.151324
min	1.000000	103.000000	1.000000	0.000006	1100.000000	1.000000
25%	26371.250000	124.000000	8.000000	0.249000	1500.000000	4.000000
50%	52736.500000	129.000000	9.000000	0.498298	2200.000000	4.000000
75%	79105.750000	148.000000	9.000000	0.749513	3700.000000	10.000000
max	105471.000000	176.000000	11.000000	0.999994	7900.000000	17.000000

8 rows × 752 columns

```
# Function to calculate missing values by column
def missing_values_table(df):
        # Total missing values
        mis_val = df.isnull().sum()

        # Percentage of missing values
        mis_val_percent = 100 * df.isnull().sum() / len(df)

        # Make a table with the results
        mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)
        mis_val_table_ren_columns = mis_val_table.rename(
        columns = {0 : 'Missing Values', 1 : '% of Total Values'})

        # Sort the table by percentage of missing descending
        mis_val_table_ren_columns = mis_val_table_ren_columns[
            mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
        '% of Total Values', ascending=False).round(1)

        print ("Your selected dataframe has " + str(df.shape[1]) + " columns.\n"
            "There are " + str(mis_val_table_ren_columns.shape[0]) +
              " columns that have missing values.")

        return mis_val_table_ren_columns
```

```
missing_values_table(data).head(50)
```

f160	18724	17.8
f159	18724	17.8
f169	18406	17.5
f170	18406	17.5
f619	18398	17.5
f618	18398	17.5
f331	18055	17.1
f330	18055	17.1
f179	17152	16.3
f180	17152	16.3
f422	14226	13.5
f653	13198	12.5
f190	12225	11.6
f189	12225	11.6
f341	11905	11.3
f340	11905	11.3
f726	11275	10.7
f664	11275	10.7
f665	11275	10.7
f666	11275	10.7
f667	11275	10.7
f668	11275	10.7
f669	11275	10.7
f640	9694	9.2
f200	9062	8.6
f199	9062	8.6
f650	8998	8.5
f651	8998	8.5
f72	8997	8.5
f586	8960	8.5
f587	8960	8.5
f649	8710	8.3
f648	8651	8.2
f588	8427	8.0
f621	8175	7.8
f620	8175	7.8

f673	7338	7.0
f672	7338	7.0
f210	6859	6.5
f209	6859	6.5
f679	6391	6.1
f150	2859	2.7
f149	2859	2.7
f32	2571	2.4

```
# Fill the missing values using 'mean'
data.fillna(data.mean(), inplace=True)
```

f202	2561	2.4

```
missing_values_table(data).head(50)
```

```
Your selected dataframe has 752 columns.
There are 0 columns that have missing values.
```

Missing Values	% of Total Values

```
data.shape
```

```
(105430, 752)
```

```
# Find all correlations and sort
correlations_data = data.corr()['loss'].sort_values()

# Print the most negative correlations
print(correlations_data.head(15), '\n')

# Print the most positive correlations
print(correlations_data.tail(15))
```

```
f612   -0.017798
f776   -0.016705
f631   -0.012730
f70    -0.010294
f69    -0.009963
f200   -0.009431
f629   -0.009031
f315   -0.008507
f1     -0.008291
f734   -0.008233
f270   -0.008216
f425   -0.008071
f428   -0.007876
f10    -0.007861
f190   -0.007687
Name: loss, dtype: float64

f674    0.020680
f536    0.026068
f471    0.042021
loss    1.000000
f33          NaN
```

```
f34          NaN
f35          NaN
f37          NaN
f38          NaN
f678         NaN
f700         NaN
f701         NaN
f702         NaN
f736         NaN
f764         NaN
Name: loss, dtype: float64
```

```
# Remove all columns with 'NaN' correlation
for i in data.columns:
    if len(set(data[i]))==1:
        data.drop(labels=[i], axis=1, inplace=True)
```

```
# Find all correlations and sort
correlations_data = data.corr()['loss'].sort_values()

# Print the most negative correlations
print(correlations_data.head(15), '\n')

# Print the most positive correlations
print(correlations_data.tail(15))
```

```
f612   -0.017798
f776   -0.016705
f631   -0.012730
f70    -0.010294
f69    -0.009963
f200   -0.009431
f629   -0.009031
f315   -0.008507
f1     -0.008291
f734   -0.008233
f270   -0.008216
f425   -0.008071
f428   -0.007876
f10    -0.007861
f190   -0.007687
Name: loss, dtype: float64

f599    0.012236
f597    0.012236
f47     0.012529
f61     0.012678
f370    0.012863
f353    0.012875
f65     0.013357
f67     0.013623
f670    0.013734
f468    0.013822
f514    0.014402
f674    0.020680
f536    0.026068
f471    0.042021
loss    1.000000
Name: loss, dtype: float64
```

```
data.shape
```

    (105430, 741)

```
def remove_collinear_features(x, threshold):
    # Dont want to remove correlations between loss
    y = x['loss']
    x = x.drop(columns = ['loss'])

    # Calculate the correlation matrix
    corr_matrix = x.corr()
    iters = range(len(corr_matrix.columns) - 1)
    drop_cols = []

    # Iterate through the correlation matrix and compare correlations
    for i in iters:
        for j in range(i):
            item = corr_matrix.iloc[j:(j+1), (i+1):(i+2)]
            col = item.columns
            row = item.index
            val = abs(item.values)

            # If correlation exceeds the threshold
            if val >= threshold:
                # Append all columns to the drop columns list
                drop_cols.append(col.values[0])

    # Drop one of each pair of correlated columns
    drops = set(drop_cols)
    x = x.drop(columns = drops)

    # Add the score back in to the data
    x['loss'] = y

    return x
```

```
data = remove_collinear_features(data, 0.6)
```

```
data.shape
```

    (105430, 155)

```
# Separate out the features and targets
features = data.drop(columns='loss')
targets = pd.DataFrame(data['loss'])

X_train, X_test, y_train, y_test = train_test_split(features, targets, test_size = 0.2, random_state

print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

    (84344, 154)
    (21086, 154)
    (84344, 1)
    (21086, 1)

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
X_train
```

```
array([[ 0.24453522,  1.38717327, -3.10669127, ..., -1.36421995,
         0.68832925, -0.670937  ],
       [ 0.94822841, -0.99089651,  1.03877486, ..., -1.34515216,
         0.80952148,  1.4904529 ],
       [ 1.55203642,  0.0962211 ,  1.03877486, ...,  0.68400886,
        -0.63282046, -0.670937  ],
       ...,
       [ 1.67240413, -0.65117226,  1.03877486, ..., -0.1909905 ,
        -0.6210486 , -0.670937  ],
       [-1.70439095, -0.311448  ,  1.03877486, ..., -0.96105474,
         0.43668168, -0.670937  ],
       [-1.21389336, -1.60240017, -1.33006293, ..., -1.17192198,
        -1.66285793,  1.4904529 ]])
```

```
# Convert y to one-dimensional array
y_train = np.array(y_train).reshape((-1, ))
y_test = np.array(y_test).reshape((-1, ))
```

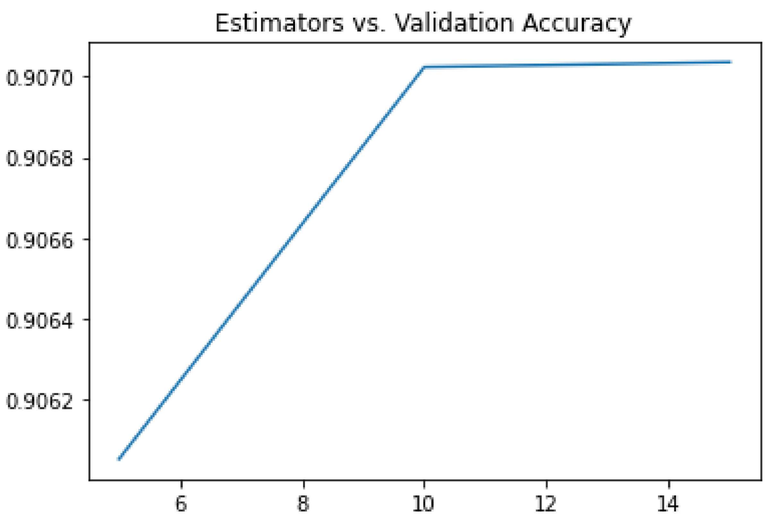
```
# Function to calculate mean absolute error
def cross_val(X_train, y_train, preds, y_test, model):
    # Applying k-Fold Cross Validation
    from sklearn.model_selection import cross_val_score
    from sklearn.metrics import accuracy_score
    val_accuracies = cross_val_score(estimator = model, X = X_train, y = y_train, cv = 5)
    accuracy = accuracy_score(y_test, preds)
    return val_accuracies.mean(), accuracy

def fit_and_evaluate(model):
    # Train the model
    model.fit(X_train, y_train)

    # Make predictions and evalute
    model_pred = model.predict(X_test)
    model_cross = cross_val(X_train, y_train, model_pred, y_test, model)

    # Return the performance metric
    return model_cross
```

```
# Random Forest Classification
from sklearn.ensemble import RandomForestClassifier
estimators = [5,10,15]
cross_accuracies = []
test_accuracies = []

for i in estimators:
  random = RandomForestClassifier(n_estimators = i, criterion = 'entropy')
  random_cross_accuracy, random_accuracy = fit_and_evaluate(random)
```

```
    cross_accuracies.append(random_cross_accuracy)
    test_accuracies.append(random_accuracy)
    print(f'Random Forest Performance for n_estimators = {i} : Validation Accuracy - {random_cross_acc
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
Random Forest Performance for n_estimators = 5 : Validation Accuracy - 0.9060514097298444 and T
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
Random Forest Performance for n_estimators = 10 : Validation Accuracy - 0.9070236177950266 and
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
Random Forest Performance for n_estimators = 15 : Validation Accuracy - 0.907035473862369 and T
```

```
import matplotlib.pyplot as plt

plt.plot(estimators, cross_accuracies)
plt.title('Estimators vs. Validation Accuracy')
plt.xlabel('No. of Estimators')
plt.ylabel('Validation Accuracy')
plt.show()
```




```
import matplotlib.pyplot as plt

plt.plot(estimators, test_accuracies)
plt.title('Estimators vs. Test Accuracy')
plt.xlabel('No. of Estimators')
plt.ylabel('Test Accuracy')
plt.show()
```

Estimators vs. Test Accuracy

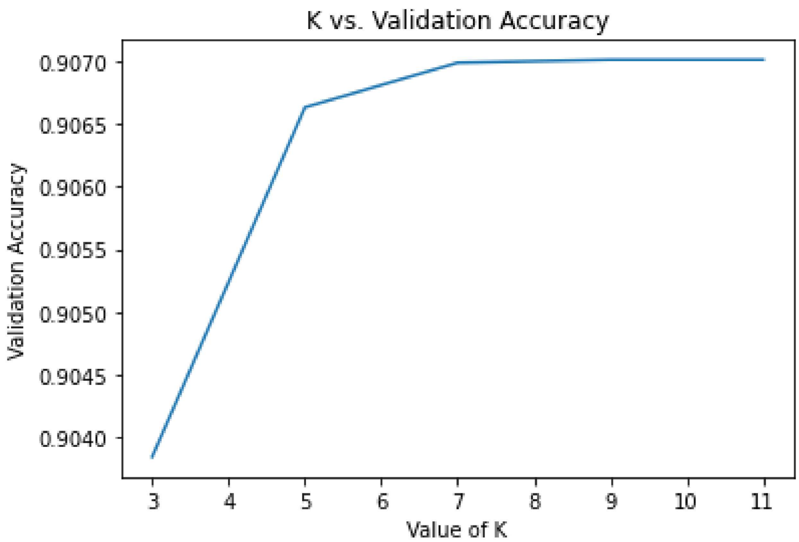
```
# Logistic Regression
from sklearn.linear_model import LogisticRegression
LR = LogisticRegression()
LR_cross_accuracy, LR_accuracy = fit_and_evaluate(LR)

print(f'Logistic Regression Performance : Validation Accuracy - {LR_cross_accuracy} and Test Accurac
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
Logistic Regression Performance : Validation Accuracy - 0.9055890132632619 and Test Accuracy -
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
# KNN
from sklearn.neighbors import KNeighborsClassifier
k = [3,5,7,9,11]
cross_accuracies_knn = []
test_accuracies_knn = []

for i in k:
  knn = KNeighborsClassifier(n_neighbors=i)
  knn_cross_accuracy, knn_accuracy = fit_and_evaluate(knn)
  cross_accuracies_knn.append(knn_cross_accuracy)
  test_accuracies_knn.append(knn_accuracy)
  print(f'KNN Performance : Validation Accuracy - {knn_cross_accuracy} and Test Accuracy - {knn_accu
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
KNN Performance : Validation Accuracy - 0.9038461446547335 and Test Accuracy - 0.906478232002227
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
KNN Performance : Validation Accuracy - 0.9066323612468652 and Test Accuracy - 0.908944323247655
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
KNN Performance : Validation Accuracy - 0.9069880481872523 and Test Accuracy - 0.909703120553922
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
KNN Performance : Validation Accuracy - 0.9070117603219373 and Test Accuracy - 0.909845395048844
/usr/local/lib/python3.7/dist-packages/sklearn/model_selection/_split.py:680: UserWarning: The
  UserWarning,
KNN Performance : Validation Accuracy - 0.9070117603219373 and Test Accuracy - 0.909892819880448
```

```
import matplotlib.pyplot as plt

plt.plot(k, cross_accuracies_knn)
plt.title('K vs. Validation Accuracy')
plt.xlabel('Value of K')
plt.ylabel('Validation Accuracy')
plt.show()
```




```
import matplotlib.pyplot as plt

plt.plot(k, test_accuracies_knn)
plt.title('k vs. Test Accuracy')
```

```
plt.xlabel('Value of K')
plt.ylabel('Test Accuracy')
plt.show()
```