Welcome to your assignment this week!

To better understand explainable AI, in this assignment, we will look at the **LIME** framework to explain potential black-box machine learning models in a model-agnostic way. We use a real-world dataset on Census income, also known as the ***Adult dataset*** available in the *UCI* ML Repository where we will predict if the potential income of people is more than $50K/year or not.

For this assignment, we will use:

- XGBoost (XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.). Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.
- Decision Tree which is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

LIME GitHub

After this assignment you will be able to: use the **LIME** framework to explain potential black-box machine learning models in a model-agnostic way.

Run the following cell to load the packages you will need.

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import gridspec

import seaborn as sns
import lime
from lime.lime_tabular import LimeTabularExplainer
import shap
import itertools

##
from utils import *
##

from sklearn.model_selection import train_test_split
from collections import Counter
import xgboost as xgb
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix

from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
```

```
import graphviz
from io import StringIO
from sklearn import datasets,tree
import pydot
import pydotplus
from sklearn.tree import export_graphviz
from IPython.display import Image


import warnings
warnings.filterwarnings('ignore')
#plt.style.use('fivethirtyeight')
%matplotlib inline

shap.initjs()
```

```
<IPython.core.display.HTML object>
```

Next, let's load the census dataset. Run the following cell to load the features `X` and the labels `y`.

```
X_raw, y = shap.datasets.adult(display=True)

X_raw = X_raw.drop(columns=['Capital Gain']) # These two features are
intentionally removed.
X_raw = X_raw.drop(columns=['Capital Loss']) # These two features are
intentionally removed.

labels = np.array([int(label) for label in y])

print('The shape of X_raw is: ',X_raw.shape)
print('The shape of y is: ',labels.shape)
```

```
The shape of X_raw is:  (32561, 10)
The shape of y is:  (32561,)
```

You've loaded:

- `X_raw`: a DataFrame containing 32,561 instances with 12 features.
- `y`: the list of binary labels for the 32,561 examples. If salary of instance i is more than \ $50K: $y^{(i)}=1$ otherwise: $y^{(i)}=0$.

Understanding the Census Income Dataset

Let's now take a look at our dataset attributes and understand their meaning and significance.

Attribute Name	Type	Description
Age	Continuous	Represents age of the person (Private, Self-emp-not-inc,

Attribute Name	Type	Description
		Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked)
Workclass	Categorical	Represents the workclass of the person. (Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked).
Education-Num	Categorical	Numeric representation of educational qualification.Ranges from 1-16.(Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc, 9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool)
Marital Status	Categorical	Represents the marital status of the person(Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse)
Occupation	Categorical	Represents the type of profession job of the person(Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces)
Relationship	Categorical	Represents the relationship status of the person(Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried)
Race	Categorical	Represents the race of the person(White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black)
Sex	Categorical	Represents the gender of the person(Female, Male)
Capital Gain	Continuous	The total capital gain for the person
Capital Loss	Continuous	The total capital loss for the person
Hours per week	Continuous	Total hours spent working per week
Country	Categorical	The country where the person was born
Income Label	Categorical	The class label column is the one we want to predict

We have a total of 12 features and our objective is to predict if the income of a person will be more than $50K (True) or less than $50K (False). Hence we will be building and interpreting a classification model.

Let's have a look at the first three instances of the dataset (you can use `X.head(3)` to see the content of the dataset):

For example, the first person is 39 years old, works for the state governement, is a Male and was born in the US. By using `y[0:3]` you can get binary values indicating whether these persons have an income higher than $50K or no.

Pre-processing

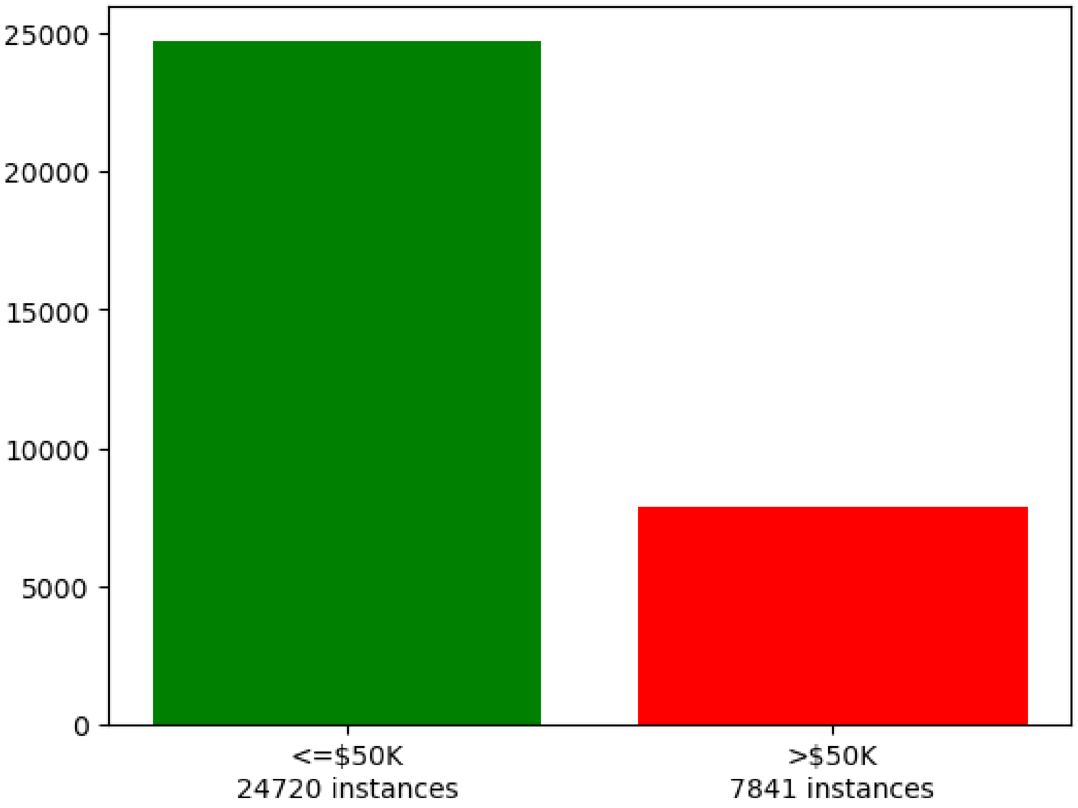
Converting the categorical columns with string values to numeric representations. Typically the XGBoost model can handle categorical data natively being a tree-based model so we don't one-hot encode the features

```
mapping = {}
cat_cols = X_raw.select_dtypes(['category']).columns
for col in cat_cols:
    mapping[col] = dict( enumerate(X_raw[col].cat.categories ) )
indices_cat_cols = [ list(X_raw.columns).index(x) for x in cat_cols ]
X = X_raw.copy()
X[cat_cols] = X_raw[cat_cols].apply(lambda x: x.cat.codes)
headers=list(X.columns)
X.head(3)
```

```
    Age  Workclass  Education-Num  Marital Status  Occupation
Relationship  \
0  39.0          7           13.0               4           1
1
1  50.0          6           13.0               2           4
0
2  38.0          4            9.0               0           6
1

   Race  Sex  Hours per week  Country
0     4    1            40.0       39
1     4    1            13.0       39
2     4    1            40.0       39
```

Let's have a look at the distribution of people with <= $50K (0) and > $50K (1) income:

```
plt.bar([0], height=[Counter(labels)[0]], color="green")
plt.bar([1], height=[Counter(labels)[1]], color="red")
plt.xticks([0, 1], ['<=$50K\n'+str(Counter(labels)[0])+' instances',
                    '>$50K\n'+str(Counter(labels)[1])+' instances'])
plt.show()
```



Split Train and Test Datasets

As in any Machine Learning, we need to partition the dataset into two subsets -- a training and testset. Please note that in practice, the dataset needs to be partitioned into three subsets, the third once being the validation set which will be used for hyperparameters tuning. However, in this assignment, we will not tune the hyperparameters.

Run the following to split the dataset accordingly:

```
X_train, X_test, y_train, y_test = train_test_split(X, labels,
test_size=0.3, random_state=42, stratify=y)
print('The shape of training set is: ',X_train.shape)
print('The shape of test set is: ',X_test.shape)

The shape of training set is:  (22792, 10)
The shape of test set is:  (9769, 10)
```

You've created:

- `X_train`: a trainig DataFrame containing 22,792 instances used for training.
- `y_train`: the list of binary labels for the 22,792 instances of the training set.
- `X_test`: a test DataFrame containing 9,769 instances used for testing.
- `y_test`: the list of binary labels for the 9,769 instances of the test set.

We note that since we are using a stratified splitting, the distribution of samples in the training and test set is similar to the distribution in the dataset, i.e., roughly 24% of positive examples in each subset.

Training the classification model

Now we train and build a boosting classification model on our training data using XGBoost (XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable.). Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees.

Run the following to start the training of the classifier:

```
xgc = xgb.XGBClassifier(n_estimators=500, max_depth=5, base_score=0.5,
                        objective='binary:logistic', random_state=42)
xgc.fit(X_train, y_train)
```

```
XGBClassifier(base_score=0.5, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None,
feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None,
importance_type=None,
              interaction_constraints=None, learning_rate=None,
max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=5, max_leaves=None,
              min_child_weight=None, missing=nan,
monotone_constraints=None,
              n_estimators=500, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=42, ...)
```

Predictions on the test data

Now that the classifier is trained, let's make few predictions on the test set:

```
predictions = xgc.predict(X_test)
print("The values predicted for the first 20 test examples are:")
print(predictions[:20])

print("The true values are:")
print(y_test[:20])
```

```
The values predicted for the first 20 test examples are:
[1 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0]
```

```
The true values are:
[1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0]
```

As you can see, our classier is making only 2 errors!

Model Performance

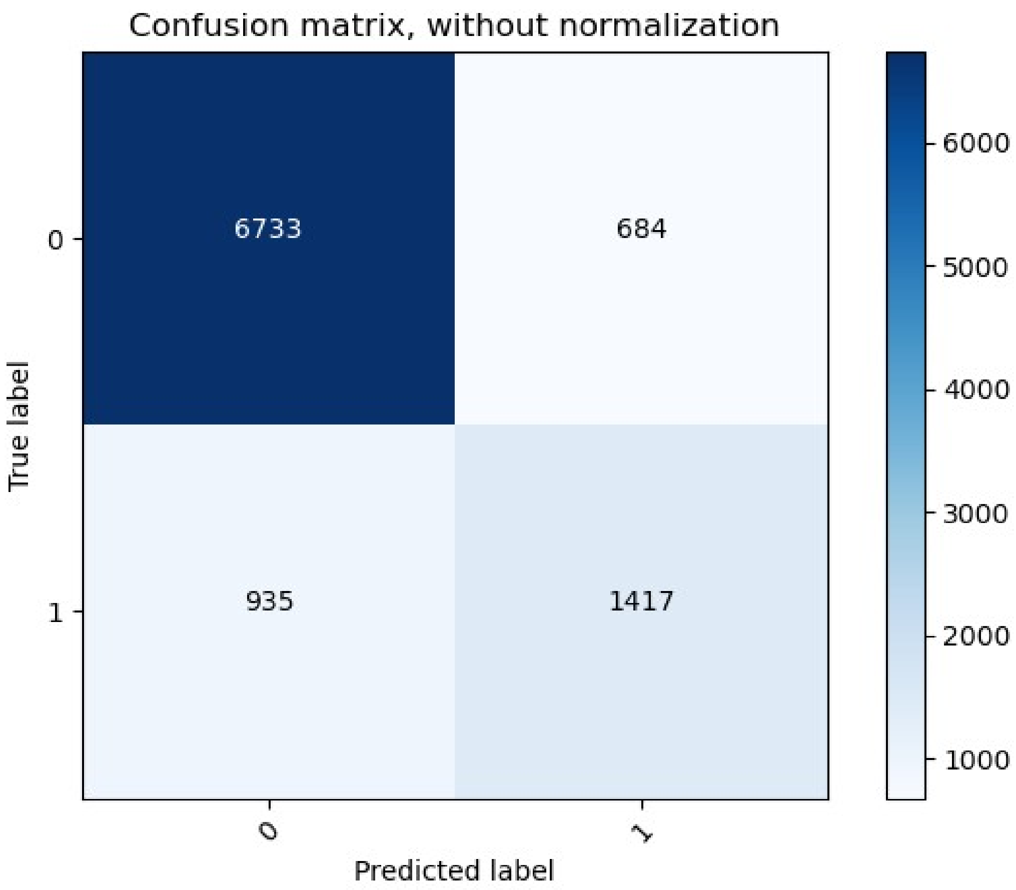
Let's now evaluate the performance of our classifier on the test set. For that, we will call `sklearn.metrics.classification_report()` which returns a text report showing the main classification metrics including: Presicion, Recall, and F1-Score. The reported averages include macro average (averaging the unweighted mean per label) and weighted average (averaging the support-weighted mean per label).

```
report = classification_report(y_test, predictions)
print(report)
```

```
              precision    recall  f1-score   support

           0       0.88      0.91      0.89      7417
           1       0.67      0.60      0.64      2352

    accuracy                           0.83      9769
   macro avg       0.78      0.76      0.76      9769
weighted avg       0.83      0.83      0.83      9769
```

To get more details, let's print the confusion matrix:

```
class_labels = list(set(labels))
cnf_matrix = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cnf_matrix, classes=class_labels,
                      title='Confusion matrix, without normalization')
```



Task 1: Please provide comments on the performance of the classifier.

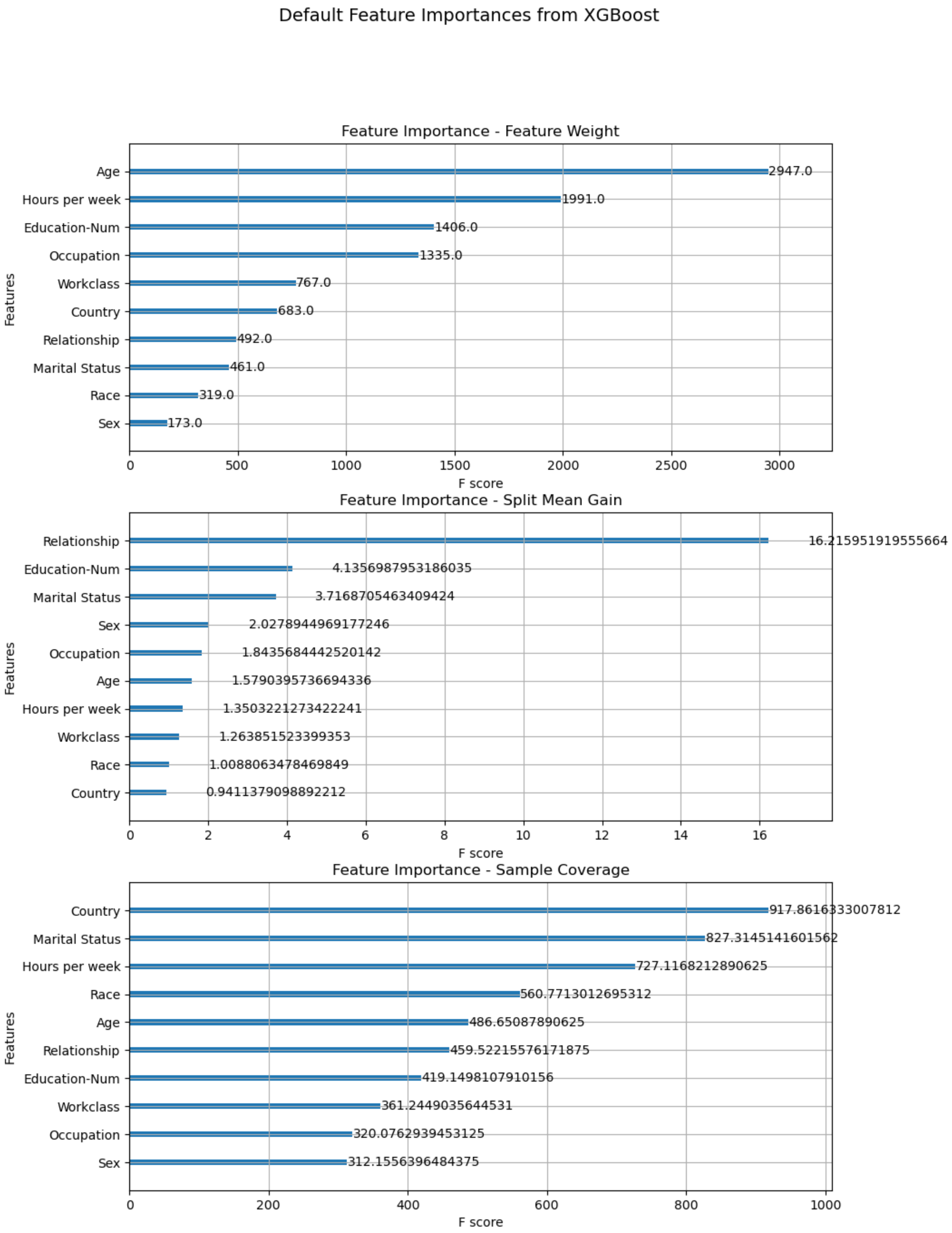
- The classifier's evaluation, based on precision, recall, and F1-score, highlights its stronger performance in distinguishing the negative class (Class 0) compared to the positive class (Class 1).
- It demonstrates a high precision of 0.88 and recall of 0.91 for Class 0, indicating its proficiency in correctly identifying instances belonging to the negative class.
- However, when it comes to the positive class, it shows lower precision (0.67) and recall (0.60), indicating difficulties in accurately recognizing positive instances and leading to more false positives and false negatives.
- Addressing this performance imbalance between the two classes might be necessary to create a more equitable and efficient model for both classes. ***

Feature importance:

The global feature importance calcuations that come with XGBoost, enables us to view feature importances based on the following:

- **Feature Weights**: This is based on the number of times a feature appears in a tree across the ensemble of trees.
- **Gain**: This is based on the average gain of splits which use the feature.
- **Coverage**: This is based on the average coverage (number of samples affected) of splits which use the feature.

```
fig = plt.figure(figsize = (10, 15))
title = fig.suptitle("Default Feature Importances from XGBoost",
fontsize=14)

ax1 = fig.add_subplot(3,1, 1)
xgb.plot_importance(xgc, importance_type='weight', ax=ax1)
t = ax1.set_title("Feature Importance - Feature Weight")

ax2 = fig.add_subplot(3,1, 2)
xgb.plot_importance(xgc, importance_type='gain', ax=ax2)
t = ax2.set_title("Feature Importance - Split Mean Gain")

ax3 = fig.add_subplot(3,1, 3)
xgb.plot_importance(xgc, importance_type='cover', ax=ax3)
t = ax3.set_title("Feature Importance - Sample Coverage")
```


Default Feature Importances from XGBoost
Feature Importance - Feature Weight
Features
Age
Hours per week
Education-Num
Occupation
Workclass
Country
Relationship
Marital Status
Race
Sex
2947.0
1991.0
1406.0
1335.0
767.0
683.0
492.0
461.0
319.0
173.0
0
500
1000
1500
2000
2500
3000
F score
Feature Importance - Split Mean Gain
Features
Relationship
Education-Num
Marital Status
Sex
Occupation
Age
Hours per week
Workclass
Race
Country
16.215951919555664
4.1356987953186035
3.7168705463409424
2.0278944969177246
1.8435684442520142
1.5790395736694336
1.3503221273422241
1.263851523399353
1.0088063478469849
0.9411379098892212
0
2
4
6
8
10
12
14
16
F score
Feature Importance - Sample Coverage
Features
Country
Marital Status
Hours per week
Race
Age
Relationship
Education-Num
Workclass
Occupation
Sex
917.8616333007812
827.3145141601562
727.1168212890625
560.7713012695312
486.65087890625
459.52215576171875
419.1498107910156
361.2449035644531
320.0762939453125
312.1556396484375
0
200
400
600
800
1000
F score

Task 2: Please provide comments on the above global feature importance calcuations.

- The calculations of global feature importance offer valuable insights into the elements that underpin the predictions made by a machine learning model.
- From these metrics, it becomes clear that Age consistently emerges as a key predictor across all three dimensions - Feature Weights, Gain, and Coverage, underscoring its central role in shaping the model's decision-making. This highlights the significance of age as a primary factor in the model's predictions.
- Similarly, Hours Per Week and Education Num also display notable importance and influence within various facets of the model's functioning.
- Simultaneously, the metrics unveil variations in the rankings of other features, emphasizing that different facets of the model may assign differing levels of importance to specific attributes.
  ***

Model Interpretation Methods

LIME:

Lime is able to explain any black box classifier, with two or more classes. All we require is that the classifier implements a function that takes in raw text or a numpy array and outputs a probability for each class. LIME tries to fit a global surrogate model, LIME focuses on fitting local surrogate models to explain why single predictions were made.

Since XGBoost has some issues with feature name ordering when building models with dataframes (we also needed feature names in the previous `plot_importance()` calls), we will build our same model with numpy arrays to make LIME work. Remember the model being built is the same ensemble model which we treat as our black box machine learning model.

Note the difference with the previous `fit` call:

```
xgc_np = xgb.XGBClassifier(n_estimators=500, max_depth=5, 
base_score=0.5,
                        objective='binary:logistic', random_state=42)
mymodel = xgc_np.fit(X_train.values, y_train)
```

`LimeTabularExplainer` class helps in explaining predictions on tabular (i.e. matrix) data. For numerical features, it perturbs them by sampling from a Normal(0,1) and doing the inverse operation of mean-centering and scaling, according to the means and stds in the training data. For categorical features, it perturbs by sampling according to the training distribution, and making a binary feature that is 1 when the value is the same as the instance being explained.

`explain_instance()` function generates explanations for a prediction. First, we generate neighborhood data by randomly perturbing features from the instance. We then learn locally weighted linear (surrogate) models on this neighborhood data to explain each of the classes in an interpretable way.

```
headers=list(X.columns)
explainer = LimeTabularExplainer(X_train.values, 
feature_names=headers, discretize_continuous=True,

categorical_features=indices_cat_cols,
                                 class_names=['<= $50K', '> 
$50K'],verbose=True)
```

When a person's income <= $50K

Lime shows which features were the most influential in the model taking the correct decision of predicting the person's income as below $50K. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

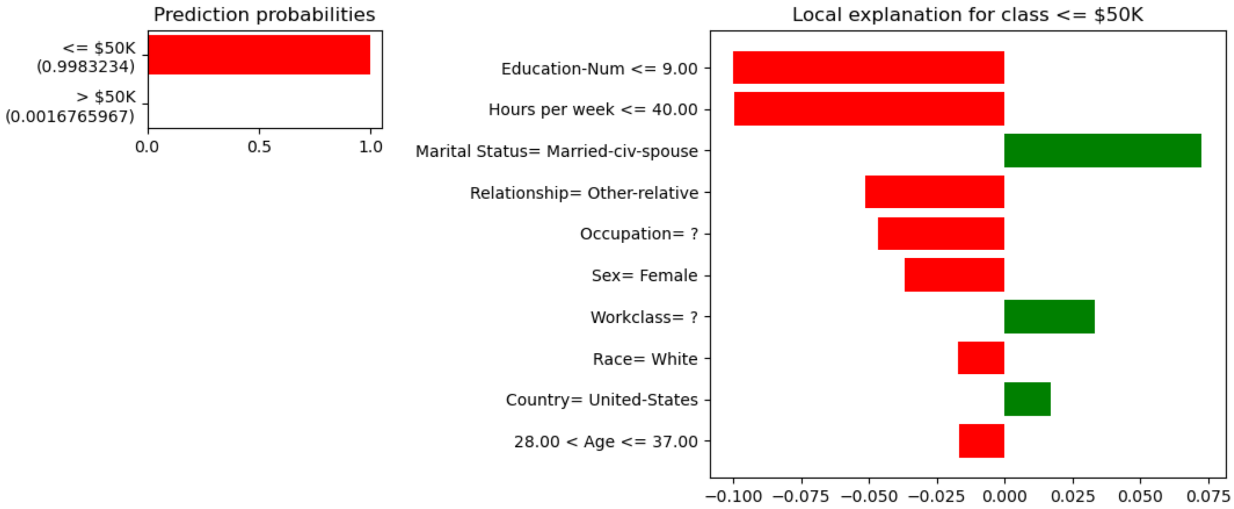
Task 3: Please find a person for which the income is <= $50K and the prediction is correct.

```
# Change only the value of to select that person:
i = 30
###########

exp1 = explainer.explain_instance(X_test.iloc[i].values, 
xgc_np.predict_proba,
                                  distance_metric='euclidean', 

num_features=len(X_test.iloc[i].values))
proba1 = xgc_np.predict_proba(X_test.values)[i]

print("********************")
print('Test id: ' , i)
print('Probability(',exp1.class_names[0],") =", exp1.predict_proba[0])
print('Probability(',exp1.class_names[1],") =", exp1.predict_proba[1])
print('Predicted Label:', predictions[i])
print('True class: ' , y_test[i])
print("********************")
```

```
Intercept 0.22513826166155745
Prediction_local [-0.02043018]
Right: 0.0016765967
********************
Test id:  30
Probability( <= $50K ) = 0.9983234
Probability( > $50K ) = 0.0016765967
Predicted Label: 0
True class:  0
********************
```

The classifier got this example right (it predicted income ¿=$50K). Let's have a look at the explanation provided by LIME:

```
plot_explanation(exp1, mapping)
```



```
<Figure size 640x480 with 0 Axes>
```

Task 4: Please provide comments on the above explanation provided by LIME.

- The LIME explanation, sheds light on income classification factors. It correctly identifies that having an Education Num of nine or less and working 40 hours per week or less significantly increase the likelihood of earning below 50K aligning with common expectations.
- Notably, it highlights that being in a married partnership (Martial Status - Married civ spouse) positively influences the probability of earning more than 50K, providing nuanced insights into marital status's income impact. It also suggests other features also play a role in income below 50K.
- This concise interpretation dissects key drivers of the model's prediction, enhancing clarity and aiding users in understanding classification reasons, while recognizing complex feature interactions. ***

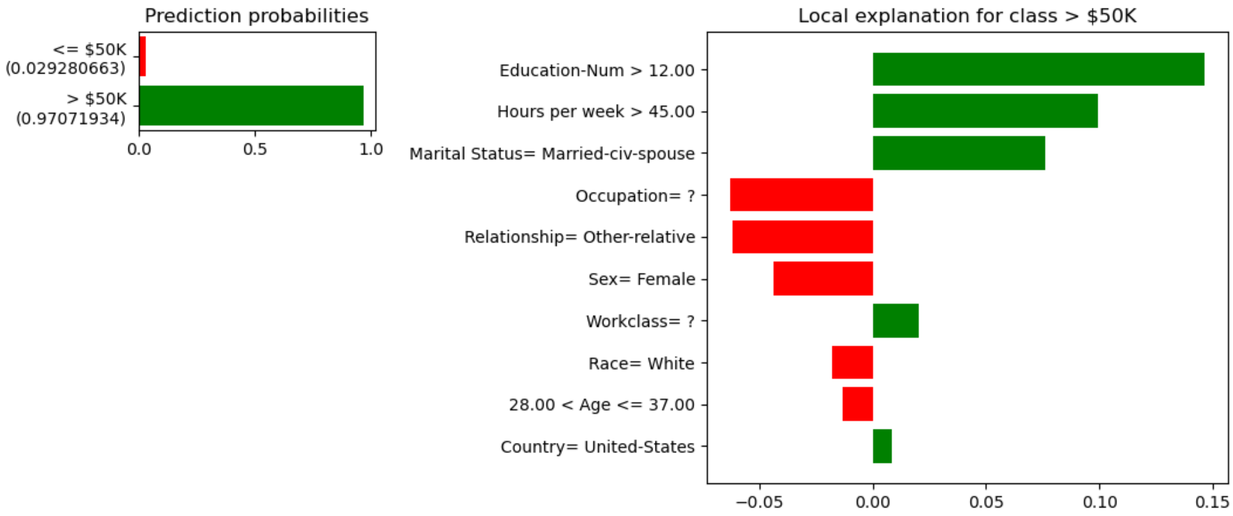
Task 5: Please change the value of one or two features the change the prediction of the classifier:

```
instanceModified1 = X_test.iloc[i]
instanceModified1['Education-Num'] = 16
instanceModified1['Hours per week'] = 50
print(instanceModified1)
```

```
Age                35.0
Workclass           0.0
Education-Num      16.0
Marital Status      2.0
Occupation          0.0
Relationship        2.0
Race                4.0
Sex                 0.0
Hours per week     50.0
Country            39.0
Name: 24579, dtype: float32
```

```
expM1 = explainer.explain_instance(instanceModified1.values,
xgc_np.predict_proba,
                                   distance_metric='euclidean',

num_features=len(instanceModified1.values))
plot_explanation(expM1, mapping)
```

```
Intercept 0.06510723066521382
Prediction_local [0.21587197]
Right: 0.97071934
```



```
<Figure size 640x480 with 0 Axes>
```

Task 6: How did you choose these features for which you have changed the value? How did you chose these values?

- The reason for choosing and changing the featues "Education-Num" to 16 and "Hours per week" to 50 was due to their significant impact on the likelihood of earning less than 50K.
- This modification was made to investigate how these changes affect income probability, specifically in scenarios where individuals possess higher educational qualifications and engage in longer working hours.

When a person's income > $50K

Lime shows which features were the most influential in the model taking the correct decision of predicting the person's income as higher $50K. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

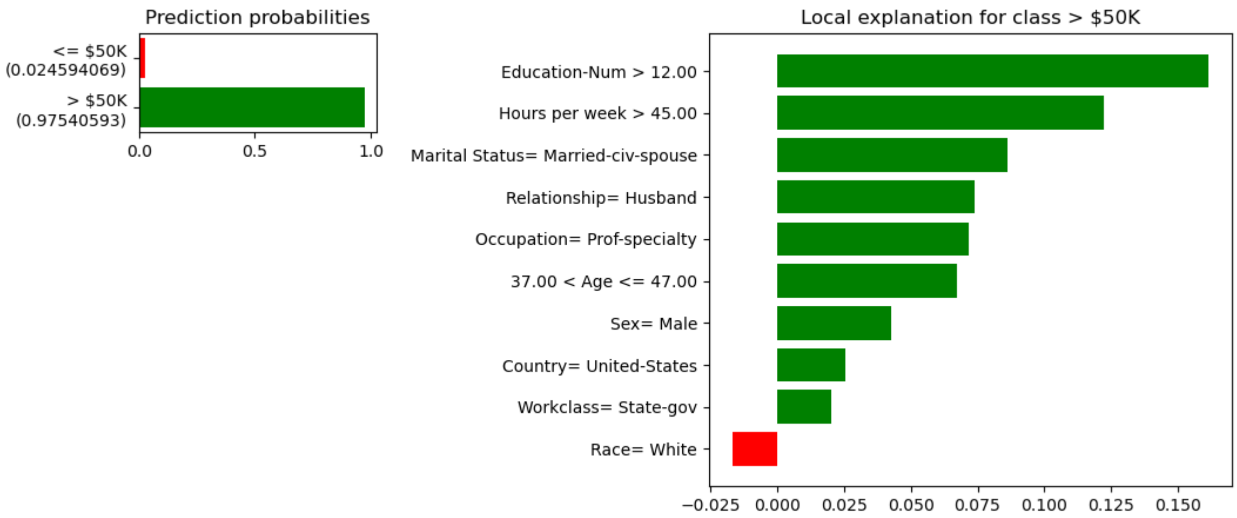
Task 7: Please find a person for which the income is > $50K and the prediction is correct.

```
# Change only the value of to select that person:
j = 21
###########

exp2 = explainer.explain_instance(X_test.iloc[j].values,
xgc_np.predict_proba,
                                  distance_metric='euclidean',

num_features=len(X_test.iloc[j].values))
proba2 = xgc_np.predict_proba(X_test.values)[j]

print("*******************")
print('Test id: ' , j)
print('Probability(',exp2.class_names[0],") =", exp2.predict_proba[0])
print('Probability(',exp2.class_names[1],") =", exp2.predict_proba[1])
print('Predicted Label:', predictions[j])
print('True class: ' , y_test[j])
print("*******************")
```

```
Intercept -0.06658677761365653
Prediction_local [0.58832129]
Right: 0.97540593
********************
Test id:  21
Probability( <= $50K ) = 0.024594069
Probability( > $50K ) = 0.97540593
Predicted Label: 1
True class:  1
********************
```

The classifier got this example right (it predicted income ¿$50K). Let's have a look at the explanation provided by LIME:

```
plot_explanation(exp2, mapping)
```



```
<Figure size 640x480 with 0 Axes>
```

Task 8: Please provide comments on the above explanation provided by LIME.

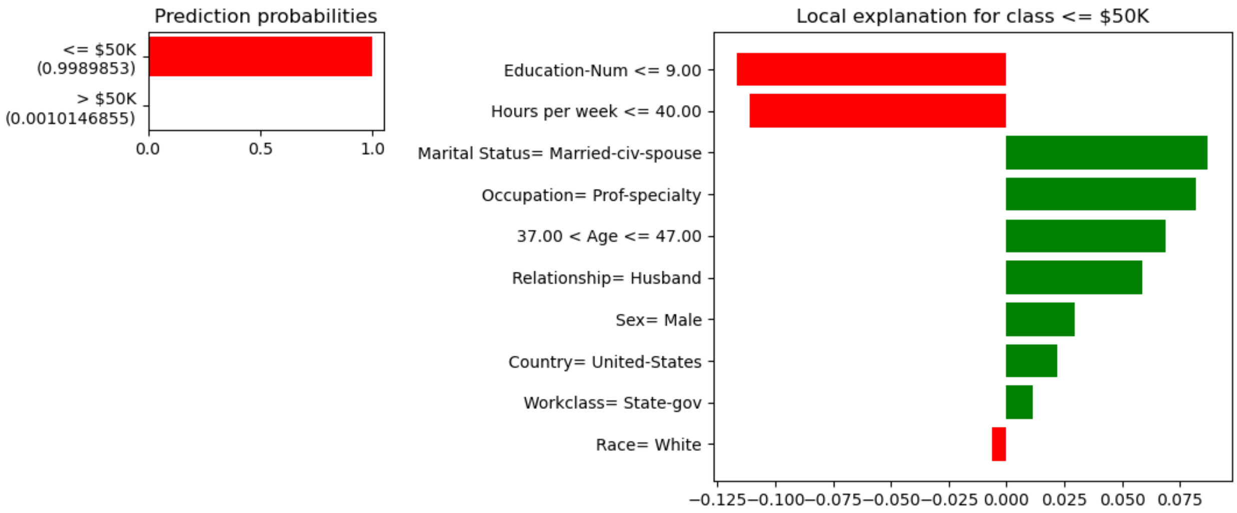
- The explanation provided by LIME for Test ID 21 is concise and informative. It indicates that the model predicted an income of greater than 50K with a high probability of 0.975, aligning with the true class label of 1.
- The explanation highlights that, for this prediction, all the features except "Race" contributed positively to the probability of having an income greater than 50K.
- Specifically, it emphasizes that "Education Num" exceeding 12, "Hours Per week" exceeding 50, and having an "Occupation" categorized as "Prof-speciality" were the top three factors influencing the positive prediction. ***

Task 9: Please change the value of one or two features the change the prediction of the classifier:

```
instanceModified2 = X_test.iloc[j]
instanceModified2['Education-Num'] = 1
instanceModified2['Hours per week'] = 10
print(instanceModified2)
```

```
Age                41.0
Workclass           7.0
Education-Num       1.0
```

```
Marital Status     2.0
Occupation        10.0
Relationship       0.0
Race               4.0
Sex                1.0
Hours per week    10.0
Country           39.0
Name: 12288, dtype: float32

expM2 = explainer.explain_instance(instanceModified2.values,
xgc_np.predict_proba,
                                   distance_metric='euclidean',

num_features=len(instanceModified2.values))
plot_explanation(expM2, mapping)

Intercept 0.13692771237548162
Prediction_local [0.26315091]
Right: 0.0010146855
```



```
<Figure size 640x480 with 0 Axes>
```

Task 10: How did you choose these features for which you have changed the value? How did you chose these values?

- The decision to select and alter the features "Education-Num" to 1 and "Hours per week" to 10 was driven by their substantial influence on the probability of earning above 50K.
- Additionally, the model made a highly confident prediction, with a probability of 0.998, that the income would be less than 50K.

- This adjustment aimed to explore the impact of these changes on income probability, particularly in scenarios where individuals have lower levels of education and work fewer hours.

When a person's income actual is different than predicted

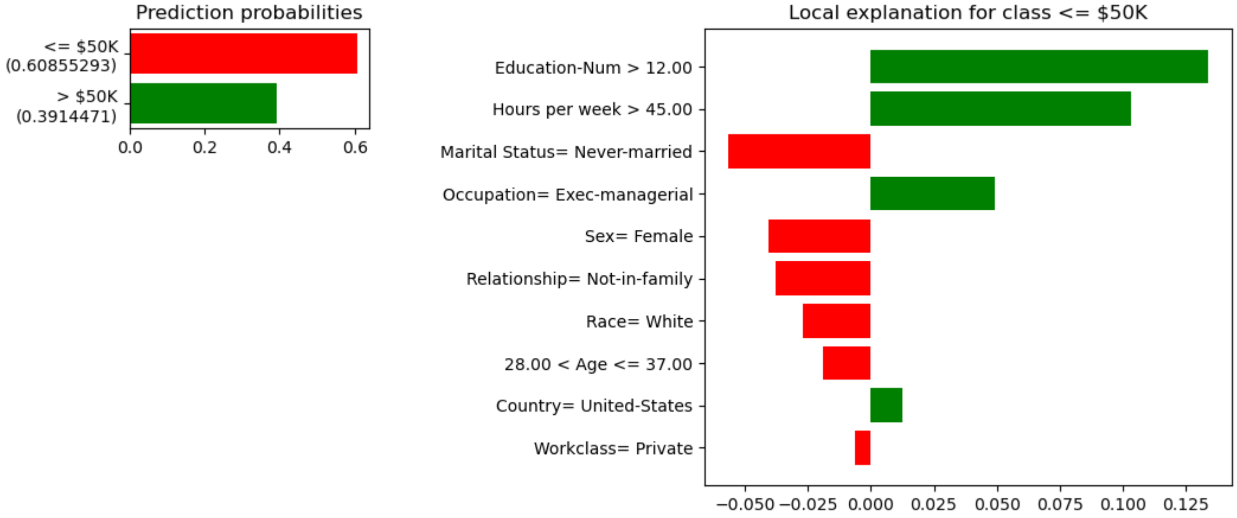
Lime shows which features were the most influential in the model taking the incorrect decision of predicting the person's income. The below explanation shows features each contributing to push the model output from the base value (the average model output over the training dataset we passed) to the actual model output. Features pushing the prediction higher are shown in red, those pushing the prediction lower are in green.

Task 11: Please find a person for which the the prediction is **incorrect**.

```
# Change only the value of to select that person:
k = 37
###########

exp3 = explainer.explain_instance(X_test.iloc[k].values,
xgc_np.predict_proba,
                                  distance_metric='euclidean',

num_features=len(X_test.iloc[k].values))
proba3 = xgc_np.predict_proba(X_test.values)[k]

print("********************")
print('Test id: ' , k)
print('Probability(',exp3.class_names[0],") =", exp3.predict_proba[0])
print('Probability(',exp3.class_names[1],") =", exp3.predict_proba[1])
print('Predicted Label:', predictions[k])
print('True class: ' , y_test[k])
print("********************")

Intercept 0.1301454505083352
Prediction_local [0.24191544]
Right: 0.3914471
********************
Test id:  37
Probability( <= $50K ) = 0.60855293
Probability( > $50K ) = 0.3914471
Predicted Label: 0
True class:  1
********************
```

The classifier got this example classified incorrectly. Let's have a look at the explanation provided by LIME:

```
plot_explanation(exp3, mapping)
```



```
<Figure size 640x480 with 0 Axes>
```

Task 12: Please provide comments on the above explanation provided by LIME.

- In Test ID 37, the model assigns a 60.86% probability of the individual having an income below 50K and a 39.14% probability of earning above 50K. However, the true class label for this instance is 1, indicating that the individual's income is, in fact, greater than 50K. This discrepancy underscores a misclassification by the model.
- The LIME explanation indicates that specific features contribute positively to the model's prediction of lower income (<= 50K). These influential features encompass "Martial Status," "Sex," "Age," and "Relationship Status," which led to misclassification.
- Furthermore, several other features, such as "Education Num," "Hours Per week," and "Occupation." are highlighted as having negative probabilities associated with income lower than 50K. ***

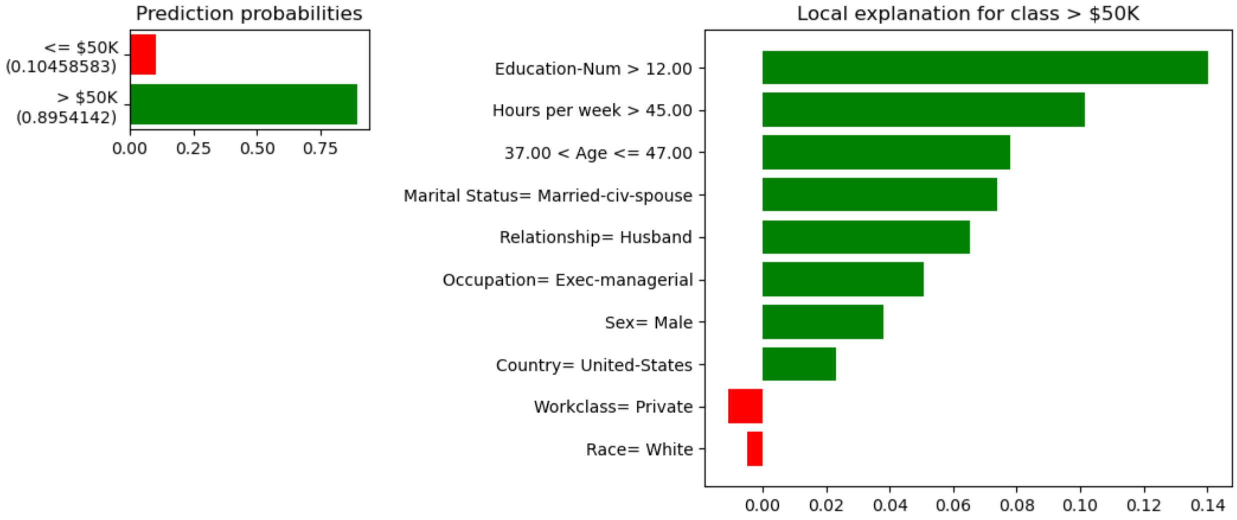
Task 13: Please change the value of one or two features the change the prediction of the classifier (to get a correct prediction):

```
instanceModified3 = X_test.iloc[k]
instanceModified3['Marital Status'] = 2
instanceModified3['Sex'] = 1
instanceModified3['Age'] = 41.0
instanceModified3['Relationship'] = 0.0

print(instanceModified3)
```

```
Age                 41.0
Workclass            4.0
Education-Num       13.0
Marital Status       2.0
Occupation           4.0
Relationship         0.0
Race                 4.0
Sex                  1.0
Hours per week      60.0
Country             39.0
Name: 31056, dtype: float32
```

```
expM3 = explainer.explain_instance(instanceModified3.values,
xgc_np.predict_proba,
                                   distance_metric='euclidean',

num_features=len(instanceModified3.values))
plot_explanation(expM3, mapping)
```

```
Intercept -0.0488035603549572
Prediction_local [0.50718611]
Right: 0.8954142
```



```
<Figure size 640x480 with 0 Axes>
```

Task 14: How did you choose these features for which you have changed the value? How did you chose these values?

- The decision to choose specific features for modification, such as "Marital Status," "Sex," "Age," and "Relationship," along with the selection of particular values, was driven by the aim of enhancing the model's predictive accuracy, in the case where the income exceeds 50K.
- These features were chosen because they were observed to have a substantial impact on the model's prediction of income being less than 50K. By modifying these features, the goal was to influence the model in a way that it would correctly predict incomes greater than 50K with a higher probability.
- For example, changing the "Relationship" as "Husband" and "Sex" as "Male" might represent different demographics or life situations where higher income is more likely. Adjusting the "Age" value could simulate scenarios where older individuals tend to earn more. Similarly, modifying the "Marital Status" feature could signify changes in family status, potentially affecting income levels.

Decision Tree

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements.

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal, but are also a popular tool in machine learning.

Let's use the DecisionTreeClassifier provided by sklearn on our dataset:

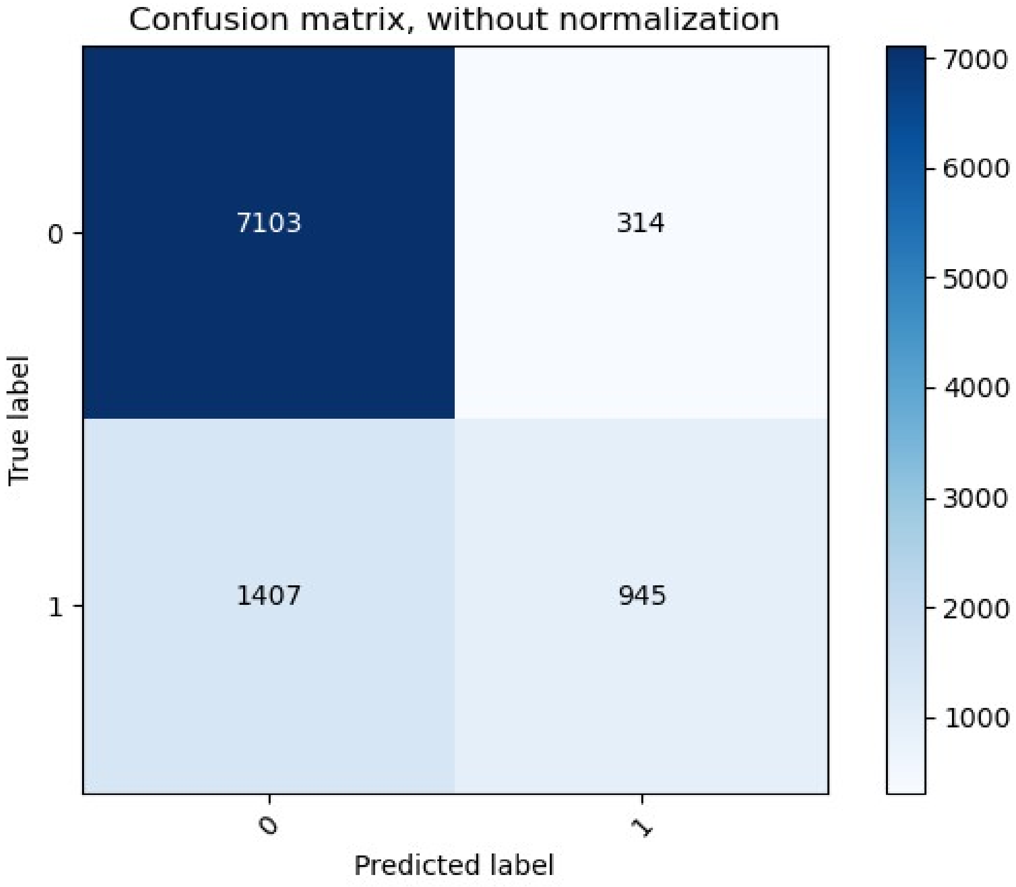
```
tree = DecisionTreeClassifier(random_state=0,
max_depth=4).fit(X_train.values, y_train)
tree.fit(X_train.values, y_train)
```

```
DecisionTreeClassifier(max_depth=4, random_state=0)
```

```
predictions = tree.predict(X_test)
print("The values predicted for the first 20 test examples are:")
print(predictions[:20])

print("The true values are:")
print(y_test[:20])
```

```
The values predicted for the first 20 test examples are:
[0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0]
```

```
The true values are:
[1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 0 0]

report = classification_report(y_test, predictions)
print(report)

              precision    recall  f1-score   support

           0       0.83      0.96      0.89      7417
           1       0.75      0.40      0.52      2352

    accuracy                           0.82      9769
   macro avg       0.79      0.68      0.71      9769
weighted avg       0.81      0.82      0.80      9769


class_labels = list(set(labels))
cnf_matrix = confusion_matrix(y_test, predictions)
plot_confusion_matrix(cnf_matrix, classes=class_labels,
                      title='Confusion matrix, without normalization')
```



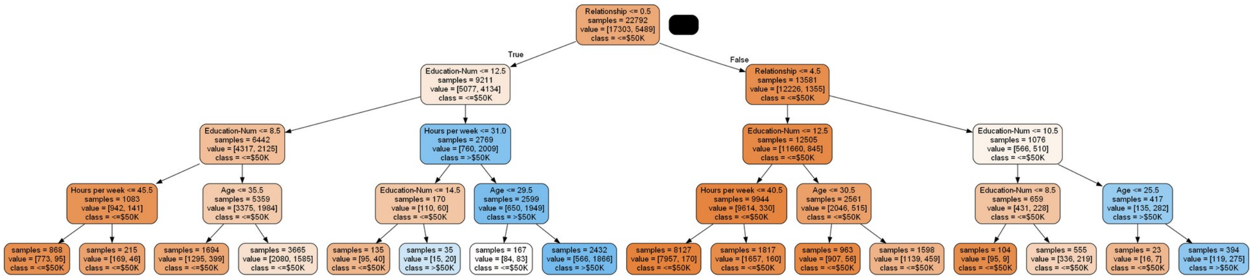
Task 15: Please provide comments on the performance of the decision Tree.

- The decision tree classifier demonstrates strong precision for Class 0 (income below 50K) at 0.83, implying its effectiveness in accurately categorizing individuals with lower incomes. However, we can see that the high recall for Class 0 at 0.96 suggests a potential bias toward this class, which results in a lower recall of 0.40 for Class 1 (income above 50K).
- The overall accuracy of 0.82 is reasonable, but it's important to acknowledge the class imbalance within the dataset, where there are likely more instances of individuals with incomes below 50K.
- The F1-score for Class 0 is robust at 0.89, indicating a balanced trade-off between precision and recall. In contrast, the F1-score for Class 1 is lower at 0.52, indicating room for improvement in accurately identifying individuals with higher incomes.

Visualzing the Tree

Let's generate a GraphViz representation of the decision tree:

```
dot_data = StringIO()
export_graphviz(tree, out_file=dot_data, feature_names=headers,
                filled=True, rounded=True, impurity= False,
class_names=['<=$50K','>$50K'])

graph=pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```



Task 16: Explain the tree structure (including the meaning of the colors).

1. Starting Point - "Relationship" Feature:

- The tree begins with an initial decision based on the "Relationship" feature. If an individual's relationship value is less than or equal to 0.50, the tree proceeds to the left subtree. Otherwise, it goes to the right subtree.

2. Left Subtree (Relationship <= 0.50):

Within the left subtree:

- The tree first evaluates the "Education-Num" feature. If "Education-Num" is less than or equal to 12.50, it goes to a further sub-level.
- If "Education-Num" is less than or equal to 8.50, the tree then considers the "Hours per week" feature.
  - If "Hours per week" is less than or equal to 45.50, the tree predicts income below 50K (class 0) with high confidence (very dark orange).
  - If "Hours per week" is greater than 45.50, the prediction remains income below 50K (class 0), but with a slightly lower confidence (dark orange).
- If "Education-Num" is greater than 8.50, the tree examines the "Age" feature.
  - If "Age" is less than or equal to 35.50, the prediction is income below 50K (class 0).
  - If "Age" is greater than 35.50, the prediction remains income below 50K (class 0).

Continuing in Left Subtree (Education-Num > 12.50):

- If "Education-Num" is greater than 12.50, the tree checks the "Hours per week" feature.
- If "Hours per week" is less than or equal to 31.00, the tree evaluates "Education-Num" once more.
  - If "Education-Num" is less than or equal to 14.50, the prediction is income below 50K (class 0).
  - If "Education-Num" is greater than 14.50, the prediction switches to income above 50K (class 1).
- If "Hours per week" is greater than 31.00, the tree examines the "Age" feature.
  - If "Age" is less than or equal to 29.50, the prediction is income below 50K (class 0).
  - If "Age" is greater than 29.50, the prediction changes to income above 50K (class 1).

3. Right Subtree (Relationship > 0.50):

In the right subtree:

- The tree again evaluates the "Relationship" feature. If it's less than or equal to 4.50, it proceeds to the left sub-level; otherwise, it goes to the right sub-level.

Continuing in Right Subtree (Education-Num <= 12.50):

- If "Education-Num" is less than or equal to 12.50 and "Hours per week" is less than or equal to 40.50, the prediction is income below 50K (class 0).

Further in Right Subtree (Education-Num > 12.50):

- If "Education-Num" is greater than 12.50, the tree checks the "Age" feature.
  - If "Age" is less than or equal to 30.50, the prediction remains income below 50K (class 0).
  - If "Age" is greater than 30.50, the prediction still stays as income below 50K (class 0).

Final Right Subtree (Relationship > 4.50):

- In the rightmost subtree, the tree focuses on the "Education-Num" feature.
- If "Education-Num" is less than or equal to 10.50, it further evaluates "Education-Num."
  - If "Education-Num" is less than or equal to 8.50, the prediction is income below 50K (class 0).
  - If "Education-Num" is greater than 8.50, the prediction remains income below 50K (class 0).
- If "Education-Num" is greater than 10.50, the tree checks the "Age" feature.
  - If "Age" is less than or equal to 25.50, the prediction is income below 50K (class 0).
  - If "Age" is greater than 25.50, the prediction switches to income above 50K (class 1).

Node Colors:

- Dark Orange: Represents predictions for income below 50K (class 0).
- Very Dark Orange: Also represents predictions for income below 50K (class 0), but with higher confidence.
- Light Orange: Represents decision nodes in the tree structure.
- White: Represents predictions for income below 50K (class 0) but with lower certainty.
- Blue: Represents predictions for income above 50K

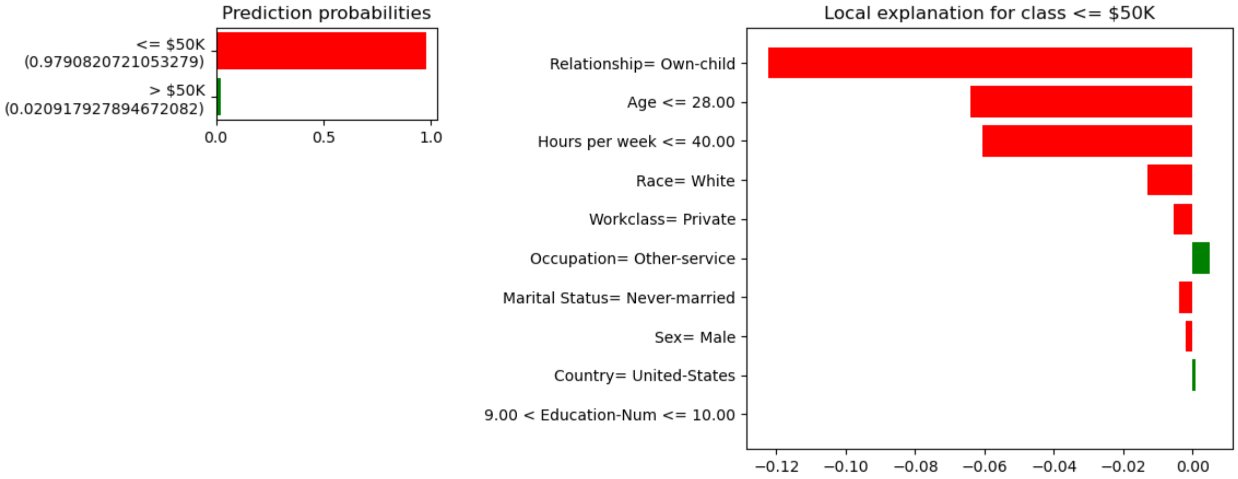
Explanation using LIME

Select any person from the dataset and get the LIME explanation for its classification.

```
# Change only the value of to select that person:
h = 100
###########

exp4 = explainer.explain_instance(X_test.iloc[h].values,
tree.predict_proba,
                                  distance_metric='euclidean',

num_features=len(X_test.iloc[h].values))
proba1 = tree.predict_proba(X_test.values)[h]

print("*******************")
print('Test id: ' , h)
print('Probability(',exp4.class_names[0],") =", exp4.predict_proba[0])
print('Probability(',exp4.class_names[1],") =", exp4.predict_proba[1])
print('Predicted Label:', predictions[h])
print('True class: ' , y_test[h])
print("*******************")
```

```
Intercept 0.27643612400892875
Prediction_local [0.01053835]
Right: 0.020917927894672082
```

```
********************
Test id:  100
Probability( <= $50K ) = 0.9790820721053279
Probability( > $50K ) = 0.020917927894672082
Predicted Label: 0
True class:  0
********************

plot_explanation(exp4, mapping)
```



```
<Figure size 640x480 with 0 Axes>
```

Task 17: Please provide comments on the above explanation provided by LIME using on the Tree structure above as a context to your explanation.

- In Test ID 100, the model predicts a high probability (97.91%) of the individual having an income below 50K and a low probability (2.09%) of earning above 50K. The predicted label is 0, which corresponds to an income below 50K, and the true class is also 0, indicating a correct prediction.

- Analyzing this prediction with respect to the decision tree structure provided earlier, we can see that the individual's features align well with the left subtree of the tree, which primarily leads to predictions of income below 50K (class 0).

- The initial decision is based on the "Relationship" feature, and in this case, "relationship - own child" falls into the category of "Relationship <= 0.50." This aligns with the left subtree.

- The tree then evaluates "Education-Num," "Age," and "Hours per week." The features "age < 28," "hours per week < 40," and "education num < 10" all match conditions within this subtree, leading to predictions of income below 50K (class 0).

Your own test example

Task 18: Following the tree above, create your own test example that will be classified as income > $\$50$K by the decision tree. Explain how you select the values for the features. Use LIME to provide explanation to that test example.

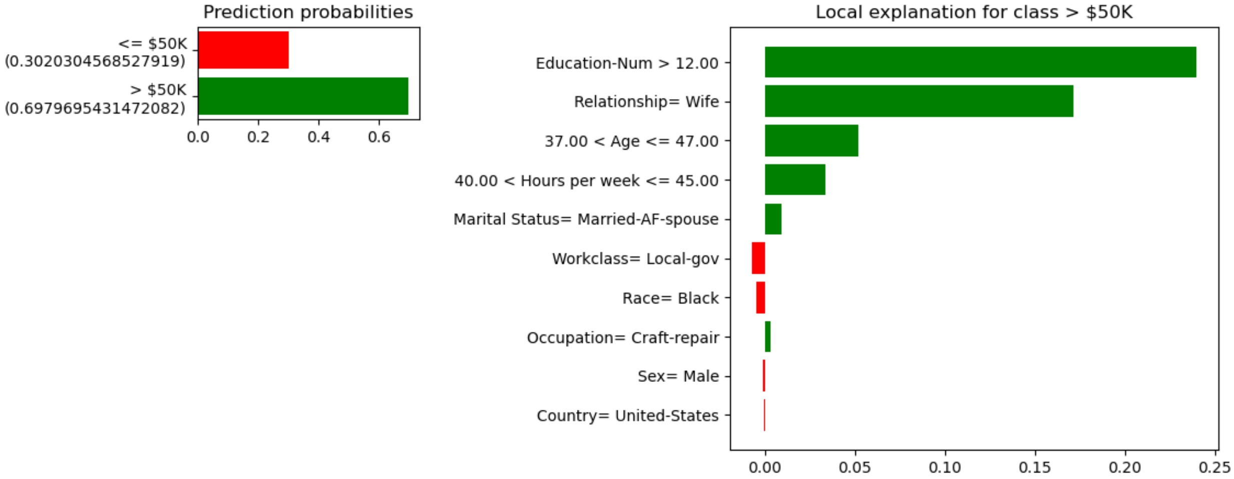
```
import pandas as pd

# Create own test example
new_row = {
    'Age': 40.0,
    'Workclass': 2,
    'Education-Num': 15.0,
    'Marital Status': 1,
    'Occupation': 3,
    'Relationship': 5,
    'Race': 2,
    'Sex': 1,
    'Hours per week': 45.0,
    'Country': 39
}

# Add the new row to X_test
X_test = pd.concat([X_test, pd.DataFrame([new_row])],
ignore_index=True)

predictions = tree.predict(X_test)

# Adding Income Label = 1 (Income > $50K)

y_test_updated = np.append(y_test, 1)
y_test_updated
```

```
array([1, 0, 0, ..., 1, 0, 1])
```

```
# Testing with our added row = 9769
p = 9769

exp5 = explainer.explain_instance(X_test.iloc[p].values,
tree.predict_proba,
                                  distance_metric='euclidean',

num_features=len(X_test.iloc[p].values))
proba2 = tree.predict_proba(X_test.values)[p]

print("********************")
print('Test id: ' , p)
print('Probability(',exp5.class_names[0],") =", exp5.predict_proba[0])
print('Probability(',exp5.class_names[1],") =", exp5.predict_proba[1])
print('Predicted Label:', predictions[p])
print('True class: ' , y_test_updated[p])
print("********************")
```

```
Intercept 0.10671915349878958
Prediction_local [0.60365244]
Right: 0.6979695431472082
********************
Test id:  9769
Probability( <= $50K ) = 0.3020304568527919
Probability( > $50K ) = 0.6979695431472082
Predicted Label: 1
True class:  1
********************
```

```
plot_explanation(exp5, mapping)
```



```
<Figure size 640x480 with 0 Axes>
```

The selection of feature values for new Test ID 9769, where the model predicts an income above 50K (class 1) with a true label of 1 and a predicted label of 1 (probability 0.69796), indicates a deliberate modification of features to achieve the desired prediction

Selection of Features and corresponding values:

- Age: 40.0 - The decision tree includes an Age threshold of 29.50. Setting Age to 40.0 ensures that the individual falls into the branch of the tree that favors a prediction of income above 50K (class 1).

- Education-Num: 15.0 - The tree indicates that an Education-Num greater than 14.50 leads to a prediction of income above 50K. By setting Education-Num to 15.0, we align with this threshold and increase the likelihood of a higher income prediction.

- Relationship: 5 - The tree structure implies that a Relationship value of 5 corresponds to being a "Wife," which is associated with a higher probability of earning above 50K.
- Hours per week: 45.0 - Hours per week feature plays a crucial role in the tree's decisions. By setting it to 45.0, we ensure that it falls into the branch of the tree where Hours per week > 31.00, favoring a prediction of higher income.
- The other features like Marital Status, Occupation, and Sex are not directly present in the provided tree structure, but they indirectly influence the predictions due to their relationships with the features present in the tree.

LIME for explaining prediction images

Task 19: Use the CIFAR100 to build an image classifier, and then use the LIME framework with visualization to explain a few predicionts. You can use any classifier for this task (Neural network, Logistic regression, etc.).

```
#Labels for CIFAR100

labels =  ['apple', 'aquarium_fish', 'baby', 'bear', 'beaver', 'bed',
'bee', 'beetle', 'bicycle', 'bottle', 'bowl', 'boy', 'bridge', 'bus',
'butterfly',
           'camel', 'can', 'castle', 'caterpillar', 'cattle', 'chair',
'chimpanzee', 'clock', 'cloud', 'cockroach', 'couch', 'crab',
'crocodile', 'cup',
           'dinosaur', 'dolphin', 'elephant', 'flatfish', 'forest',
'fox', 'girl', 'hamster', 'house', 'kangaroo', 'computer_keyboard',
           'lamp', 'lawn_mower', 'leopard', 'lion', 'lizard',
'lobster', 'man', 'maple_tree', 'motorcycle', 'mountain', 'mouse',
'mushroom',
           'oak_tree', 'orange', 'orchid', 'otter', 'palm_tree',
'pear', 'pickup_truck', 'pine_tree', 'plain', 'plate', 'poppy',
'porcupine', 'possum',
           'rabbit', 'raccoon', 'ray', 'road', 'rocket', 'rose',
           'sea', 'seal', 'shark', 'shrew', 'skunk', 'skyscraper',
'snail', 'snake', 'spider', 'squirrel', 'streetcar', 'sunflower',
'sweet_pepper',
           'table', 'tank', 'telephone', 'television', 'tiger',
'tractor', 'train', 'trout', 'tulip', 'turtle',
           'wardrobe', 'whale', 'willow_tree', 'wolf', 'woman',
'worm']

import os
import sys
import io
import tensorflow as tf
from skimage.segmentation import mark_boundaries, quickshift
from tensorflow.keras.datasets import cifar100
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from lime.lime_image import LimeImageExplainer


#Define a function to temporarily suppress output
def suppress_output():
    sys.stdout = io.StringIO()

#Define a function to restore standard output
def restore_output():
    sys.stdout.close()
    sys.stdout = sys.__stdout__

# Load and Preprocess the CIFAR-100 Dataset
(train_images, train_labels), (test_images, test_labels) =
cifar100.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

#Build and Train a CNN Classifier
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    Flatten(),
    Dense(64, activation='relu'),
    #Dropout(0.5),
    Dense(100, activation='softmax')
])

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(train_images, train_labels, epochs=100,
validation_data=(test_images, test_labels))


# Use LIME to Explain Predictions
def predict_fn(images):
    return model.predict(images)

# Create a LIME explainer for 5 Predictions
explainer = LimeImageExplainer()

num_images_to_explain = 5
```

```
sample_indices = np.random.choice(test_images.shape[0],
num_images_to_explain, replace=False)
sample_images = test_images[sample_indices]
sample_labels = test_labels[sample_indices]

# Set the figure size for larger images
plt.figure(figsize=(12, 12))


for i in range(num_images_to_explain):
    suppress_output()
    explanation = explainer.explain_instance(sample_images[i],
predict_fn, top_labels=5, segmentation_fn=quickshift)
    restore_output()

    # Get the true label and top predicted label
    true_label = labels[sample_labels[i][0]]
    top_label = labels[explanation.top_labels[0]]


    # Display the original image with true label
    plt.subplot(5, 2, i * 2 + 1)
    plt.imshow(sample_images[i])
    plt.title(f"True Label: {sample_labels[i][0]} ({true_label})")


    # Display the LIME explanation with predicted label
    temp, mask =
explanation.get_image_and_mask(explanation.top_labels[0],
positive_only=True, num_features=10, hide_rest=True)

    plt.subplot(5, 2, i * 2 + 2)
    plt.imshow(mark_boundaries(temp, mask))
    plt.title(f"LIME Explanation: {explanation.top_labels[0]}
({top_label})")


plt.tight_layout()
plt.show()

# Restore standard output at the end of the cell
restore_output()
```

```
Epoch 1/100
1563/1563 [==============================] - 31s 19ms/step - loss:
3.9368 - accuracy: 0.0952 - val_loss: 3.4913 - val_accuracy: 0.1731
Epoch 2/100
1563/1563 [==============================] - 28s 18ms/step - loss:
3.2562 - accuracy: 0.2091 - val_loss: 3.1237 - val_accuracy: 0.2427
Epoch 3/100
```

```
1563/1563 [==============================] - 28s 18ms/step - loss:
2.9551 - accuracy: 0.2642 - val_loss: 2.8753 - val_accuracy: 0.2860
Epoch 4/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.7600 - accuracy: 0.3066 - val_loss: 2.8273 - val_accuracy: 0.2950
Epoch 5/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.6155 - accuracy: 0.3343 - val_loss: 2.7247 - val_accuracy: 0.3212
Epoch 6/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.5067 - accuracy: 0.3574 - val_loss: 2.7024 - val_accuracy: 0.3292
Epoch 7/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.4159 - accuracy: 0.3764 - val_loss: 2.6098 - val_accuracy: 0.3493
Epoch 8/100
1563/1563 [==============================] - 30s 19ms/step - loss:
2.3427 - accuracy: 0.3929 - val_loss: 2.5991 - val_accuracy: 0.3483
Epoch 9/100
1563/1563 [==============================] - 30s 19ms/step - loss:
2.2727 - accuracy: 0.4067 - val_loss: 2.5736 - val_accuracy: 0.3531
Epoch 10/100
1563/1563 [==============================] - 30s 19ms/step - loss:
2.2063 - accuracy: 0.4202 - val_loss: 2.5302 - val_accuracy: 0.3616
Epoch 11/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.1524 - accuracy: 0.4316 - val_loss: 2.5175 - val_accuracy: 0.3656
Epoch 12/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.1016 - accuracy: 0.4419 - val_loss: 2.5171 - val_accuracy: 0.3678
Epoch 13/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.0564 - accuracy: 0.4525 - val_loss: 2.5471 - val_accuracy: 0.3653
Epoch 14/100
1563/1563 [==============================] - 29s 19ms/step - loss:
2.0035 - accuracy: 0.4643 - val_loss: 2.5448 - val_accuracy: 0.3591
Epoch 15/100
1563/1563 [==============================] - 29s 19ms/step - loss:
1.9634 - accuracy: 0.4720 - val_loss: 2.5907 - val_accuracy: 0.3593
Epoch 16/100
1563/1563 [==============================] - 29s 19ms/step - loss:
1.9158 - accuracy: 0.4828 - val_loss: 2.5838 - val_accuracy: 0.3606
Epoch 17/100
1563/1563 [==============================] - 30s 19ms/step - loss:
1.8792 - accuracy: 0.4903 - val_loss: 2.6509 - val_accuracy: 0.3550
Epoch 18/100
1563/1563 [==============================] - 29s 19ms/step - loss:
1.8471 - accuracy: 0.4977 - val_loss: 2.5995 - val_accuracy: 0.3686
Epoch 19/100
1563/1563 [==============================] - 30s 19ms/step - loss:
```

```
1.8114 - accuracy: 0.5037 - val_loss: 2.6458 - val_accuracy: 0.3606
Epoch 20/100
1563/1563 [==============================] - 31s 20ms/step - loss:
1.7776 - accuracy: 0.5122 - val_loss: 2.6436 - val_accuracy: 0.3615
Epoch 21/100
1563/1563 [==============================] - 30s 19ms/step - loss:
1.7461 - accuracy: 0.5175 - val_loss: 2.6880 - val_accuracy: 0.3546
Epoch 22/100
1563/1563 [==============================] - 30s 19ms/step - loss:
1.7206 - accuracy: 0.5244 - val_loss: 2.7874 - val_accuracy: 0.3450
Epoch 23/100
1563/1563 [==============================] - 30s 19ms/step - loss:
1.6874 - accuracy: 0.5309 - val_loss: 2.8009 - val_accuracy: 0.3529
Epoch 24/100
1563/1563 [==============================] - 28s 18ms/step - loss:
1.6576 - accuracy: 0.5373 - val_loss: 2.7807 - val_accuracy: 0.3564
Epoch 25/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.6291 - accuracy: 0.5414 - val_loss: 2.8320 - val_accuracy: 0.3522
Epoch 26/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.6128 - accuracy: 0.5453 - val_loss: 2.8866 - val_accuracy: 0.3484
Epoch 27/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.5787 - accuracy: 0.5561 - val_loss: 2.8891 - val_accuracy: 0.3462
Epoch 28/100
1563/1563 [==============================] - 24s 16ms/step - loss:
1.5608 - accuracy: 0.5582 - val_loss: 2.9488 - val_accuracy: 0.3510
Epoch 29/100
1563/1563 [==============================] - 25s 16ms/step - loss:
1.5387 - accuracy: 0.5627 - val_loss: 2.9417 - val_accuracy: 0.3453
Epoch 30/100
1563/1563 [==============================] - 25s 16ms/step - loss:
1.5139 - accuracy: 0.5678 - val_loss: 2.9999 - val_accuracy: 0.3423
Epoch 31/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.4927 - accuracy: 0.5730 - val_loss: 3.0710 - val_accuracy: 0.3350
Epoch 32/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.4727 - accuracy: 0.5807 - val_loss: 3.0333 - val_accuracy: 0.3448
Epoch 33/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.4531 - accuracy: 0.5858 - val_loss: 3.1463 - val_accuracy: 0.3356
Epoch 34/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.4309 - accuracy: 0.5892 - val_loss: 3.1668 - val_accuracy: 0.3359
Epoch 35/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.4123 - accuracy: 0.5928 - val_loss: 3.1704 - val_accuracy: 0.3409
```

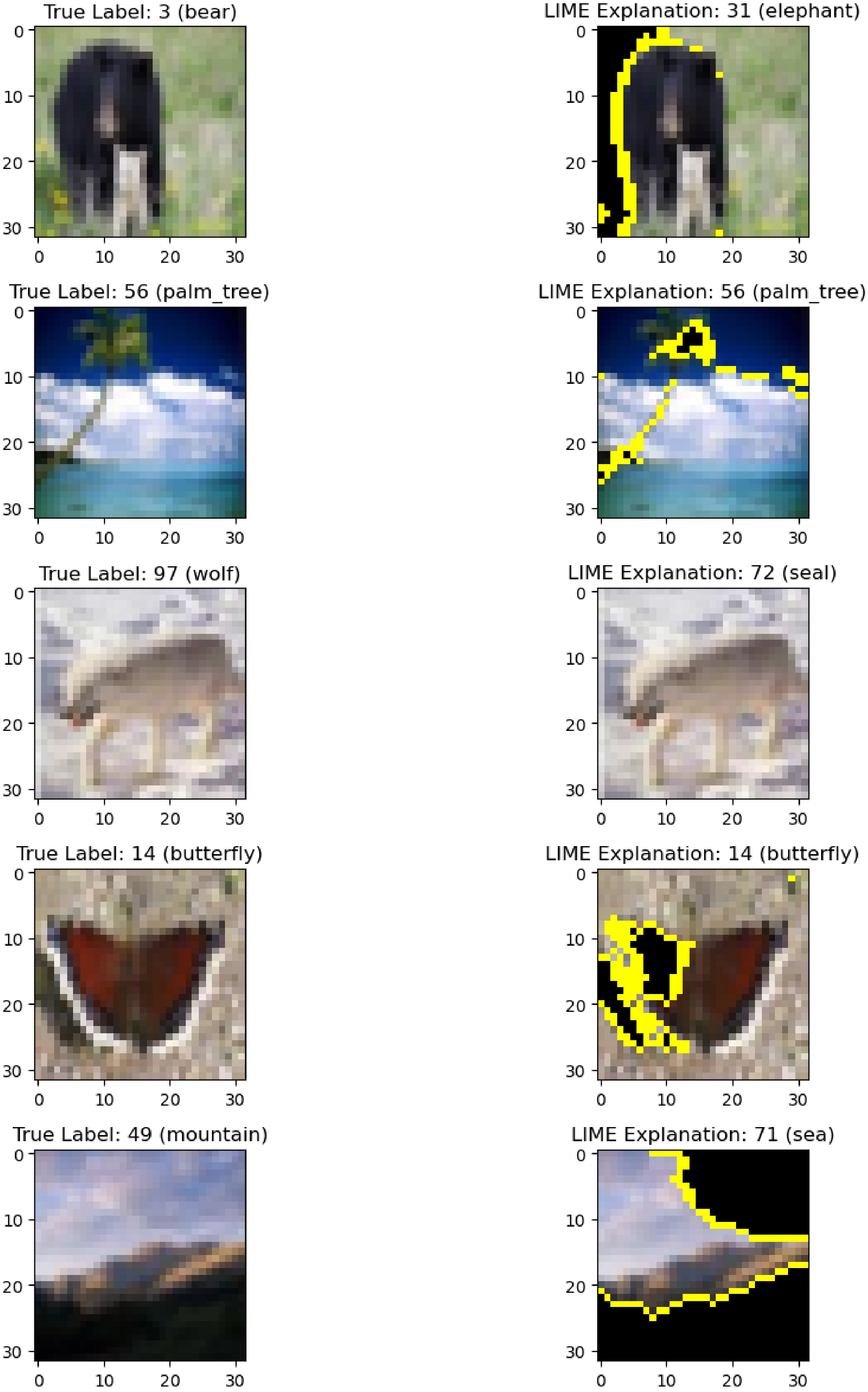
```
Epoch 36/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.3983 - accuracy: 0.5987 - val_loss: 3.1901 - val_accuracy: 0.3429
Epoch 37/100
1563/1563 [==============================] - 26s 17ms/step - loss:
1.3721 - accuracy: 0.6011 - val_loss: 3.3024 - val_accuracy: 0.3281
Epoch 38/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.3565 - accuracy: 0.6076 - val_loss: 3.3281 - val_accuracy: 0.3336
Epoch 39/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.3340 - accuracy: 0.6131 - val_loss: 3.2941 - val_accuracy: 0.3339
Epoch 40/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.3187 - accuracy: 0.6201 - val_loss: 3.3991 - val_accuracy: 0.3312
Epoch 41/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2926 - accuracy: 0.6232 - val_loss: 3.4677 - val_accuracy: 0.3284
Epoch 42/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2884 - accuracy: 0.6234 - val_loss: 3.4695 - val_accuracy: 0.3258
Epoch 43/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2698 - accuracy: 0.6287 - val_loss: 3.5250 - val_accuracy: 0.3269
Epoch 44/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2469 - accuracy: 0.6339 - val_loss: 3.6288 - val_accuracy: 0.3231
Epoch 45/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2363 - accuracy: 0.6350 - val_loss: 3.6884 - val_accuracy: 0.3219
Epoch 46/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2262 - accuracy: 0.6379 - val_loss: 3.6381 - val_accuracy: 0.3249
Epoch 47/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.2026 - accuracy: 0.6460 - val_loss: 3.7811 - val_accuracy: 0.3196
Epoch 48/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1945 - accuracy: 0.6471 - val_loss: 3.7178 - val_accuracy: 0.3236
Epoch 49/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1817 - accuracy: 0.6504 - val_loss: 3.8404 - val_accuracy: 0.3209
Epoch 50/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1685 - accuracy: 0.6544 - val_loss: 3.9186 - val_accuracy: 0.3162
Epoch 51/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1530 - accuracy: 0.6596 - val_loss: 3.8841 - val_accuracy: 0.3162
Epoch 52/100
```

```
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1529 - accuracy: 0.6575 - val_loss: 4.0250 - val_accuracy: 0.3174
Epoch 53/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1257 - accuracy: 0.6636 - val_loss: 4.0008 - val_accuracy: 0.3101
Epoch 54/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1255 - accuracy: 0.6638 - val_loss: 4.0452 - val_accuracy: 0.3210
Epoch 55/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.1115 - accuracy: 0.6682 - val_loss: 4.1156 - val_accuracy: 0.3096
Epoch 56/100

1563/1563 [==============================] - 27s 17ms/step - loss:
1.0964 - accuracy: 0.6728 - val_loss: 4.2011 - val_accuracy: 0.3075
Epoch 57/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0918 - accuracy: 0.6724 - val_loss: 4.2174 - val_accuracy: 0.3087
Epoch 58/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0801 - accuracy: 0.6759 - val_loss: 4.2290 - val_accuracy: 0.3124
Epoch 59/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0685 - accuracy: 0.6786 - val_loss: 4.2705 - val_accuracy: 0.3121
Epoch 60/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0567 - accuracy: 0.6804 - val_loss: 4.3269 - val_accuracy: 0.3016
Epoch 61/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0575 - accuracy: 0.6817 - val_loss: 4.3698 - val_accuracy: 0.3110
Epoch 62/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0453 - accuracy: 0.6860 - val_loss: 4.3500 - val_accuracy: 0.3119
Epoch 63/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0470 - accuracy: 0.6840 - val_loss: 4.4926 - val_accuracy: 0.3151
Epoch 64/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0359 - accuracy: 0.6853 - val_loss: 4.4951 - val_accuracy: 0.3057
Epoch 65/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0229 - accuracy: 0.6896 - val_loss: 4.5489 - val_accuracy: 0.3029
Epoch 66/100
1563/1563 [==============================] - 27s 18ms/step - loss:
1.0192 - accuracy: 0.6900 - val_loss: 4.5687 - val_accuracy: 0.3073
Epoch 67/100
1563/1563 [==============================] - 27s 17ms/step - loss:
1.0157 - accuracy: 0.6907 - val_loss: 4.6508 - val_accuracy: 0.3100
Epoch 68/100
1563/1563 [==============================] - 26s 17ms/step - loss:
```

```
1.0025 - accuracy: 0.6945 - val_loss: 4.6058 - val_accuracy: 0.3005
Epoch 69/100
1563/1563 [==============================] - 26s 17ms/step - loss:
0.9981 - accuracy: 0.6960 - val_loss: 4.6908 - val_accuracy: 0.3026
Epoch 70/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9884 - accuracy: 0.7018 - val_loss: 4.6669 - val_accuracy: 0.2984
Epoch 71/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9812 - accuracy: 0.7004 - val_loss: 4.7407 - val_accuracy: 0.3014
Epoch 72/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9740 - accuracy: 0.7019 - val_loss: 4.8228 - val_accuracy: 0.3074
Epoch 73/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9682 - accuracy: 0.7045 - val_loss: 4.9231 - val_accuracy: 0.3029
Epoch 74/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9570 - accuracy: 0.7069 - val_loss: 4.9282 - val_accuracy: 0.3028
Epoch 75/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9592 - accuracy: 0.7066 - val_loss: 5.0853 - val_accuracy: 0.2945
Epoch 76/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9568 - accuracy: 0.7084 - val_loss: 4.9691 - val_accuracy: 0.2965
Epoch 77/100
1563/1563 [==============================] - 27s 17ms/step - loss:
0.9492 - accuracy: 0.7064 - val_loss: 5.0682 - val_accuracy: 0.2960
Epoch 78/100
1563/1563 [==============================] - 26s 17ms/step - loss:
0.9434 - accuracy: 0.7117 - val_loss: 5.0709 - val_accuracy: 0.3009
Epoch 79/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9238 - accuracy: 0.7148 - val_loss: 5.1665 - val_accuracy: 0.3001
Epoch 80/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9400 - accuracy: 0.7103 - val_loss: 5.0970 - val_accuracy: 0.2980
Epoch 81/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9312 - accuracy: 0.7134 - val_loss: 5.1148 - val_accuracy: 0.2939
Epoch 82/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9165 - accuracy: 0.7181 - val_loss: 5.3013 - val_accuracy: 0.2935
Epoch 83/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9242 - accuracy: 0.7160 - val_loss: 5.4158 - val_accuracy: 0.2937
Epoch 84/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9099 - accuracy: 0.7193 - val_loss: 5.2450 - val_accuracy: 0.3000
```

```
Epoch 85/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9040 - accuracy: 0.7203 - val_loss: 5.4089 - val_accuracy: 0.2955
Epoch 86/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9115 - accuracy: 0.7183 - val_loss: 5.3156 - val_accuracy: 0.2980
Epoch 87/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.9050 - accuracy: 0.7209 - val_loss: 5.4602 - val_accuracy: 0.2908
Epoch 88/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8994 - accuracy: 0.7219 - val_loss: 5.4760 - val_accuracy: 0.2921
Epoch 89/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8863 - accuracy: 0.7248 - val_loss: 5.4660 - val_accuracy: 0.2972
Epoch 90/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8929 - accuracy: 0.7253 - val_loss: 5.6279 - val_accuracy: 0.2855
Epoch 91/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8878 - accuracy: 0.7231 - val_loss: 5.6314 - val_accuracy: 0.2884
Epoch 92/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8830 - accuracy: 0.7276 - val_loss: 5.6237 - val_accuracy: 0.2910
Epoch 93/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8806 - accuracy: 0.7262 - val_loss: 5.7845 - val_accuracy: 0.2946
Epoch 94/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8705 - accuracy: 0.7286 - val_loss: 5.7511 - val_accuracy: 0.2862
Epoch 95/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8755 - accuracy: 0.7295 - val_loss: 5.7848 - val_accuracy: 0.2945
Epoch 96/100
1563/1563 [==============================] - 26s 16ms/step - loss:
0.8601 - accuracy: 0.7312 - val_loss: 5.7839 - val_accuracy: 0.2914
Epoch 97/100
1563/1563 [==============================] - 26s 16ms/step - loss:
0.8528 - accuracy: 0.7354 - val_loss: 5.7533 - val_accuracy: 0.2902
Epoch 98/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8634 - accuracy: 0.7309 - val_loss: 5.9218 - val_accuracy: 0.2854
Epoch 99/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8468 - accuracy: 0.7355 - val_loss: 5.8667 - val_accuracy: 0.2844
Epoch 100/100
1563/1563 [==============================] - 25s 16ms/step - loss:
0.8529 - accuracy: 0.7353 - val_loss: 5.9873 - val_accuracy: 0.2880
```

```
{"model_id":"44ed902d6fbb457094b54357f268d906","version_major":2,"version_minor":0}

{"model_id":"bf3e66d3cf9b4dd4815cc50be35ad231","version_major":2,"version_minor":0}

{"model_id":"b253b535d2364b9d8cfa5f9931e03ecc","version_major":2,"version_minor":0}

{"model_id":"2b8f70007c3e4666bd4611e7174dec00","version_major":2,"version_minor":0}

{"model_id":"4ab48356a45e49ed88eed75f72dae352","version_major":2,"version_minor":0}
```



True Label: 3 (bear)

LIME Explanation: 31 (elephant)

True Label: 56 (palm_tree)

LIME Explanation: 56 (palm_tree)

True Label: 97 (wolf)

LIME Explanation: 72 (seal)

True Label: 14 (butterfly)

LIME Explanation: 14 (butterfly)

True Label: 49 (mountain)

LIME Explanation: 71 (sea)

LIME Explanation:

- LIME generates an explanation that highlights the most influential regions of the image for the model's prediction.
- These influential regions are visually highlighted in the image, helping users understand which parts of the image contributed most to the prediction.
- The output visualization combines the original image, its true label, and the LIME-generated explanation for easy interpretation.
- Additionaly,the simple CNN model used in this code provides a quick and accessible way to demonstrate LIME explanations, achieving higher accuracy on challenging datasets may require the use of pre-trained models like VGG16 or RESNET50, which can be computationally intensive and time-consuming to train.

Congratulations!

You've come to the end of this assignment, and have seen a lot of the ways to explain the predictions given by a classifier.

Congratulations on finishing this notebook!