AutoGluon Assignment

❖ **IEEE Fraud Detection**

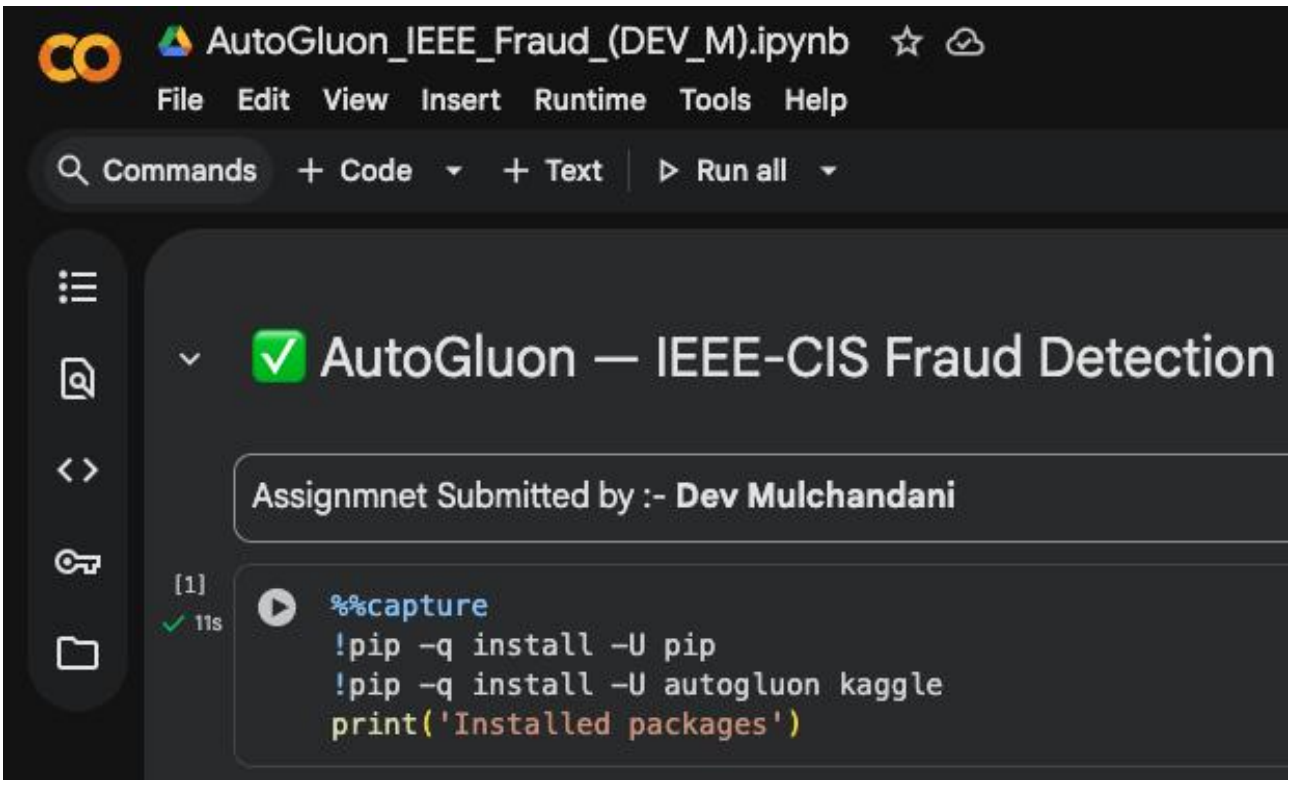
❖ **Colab Notebook :-** Link

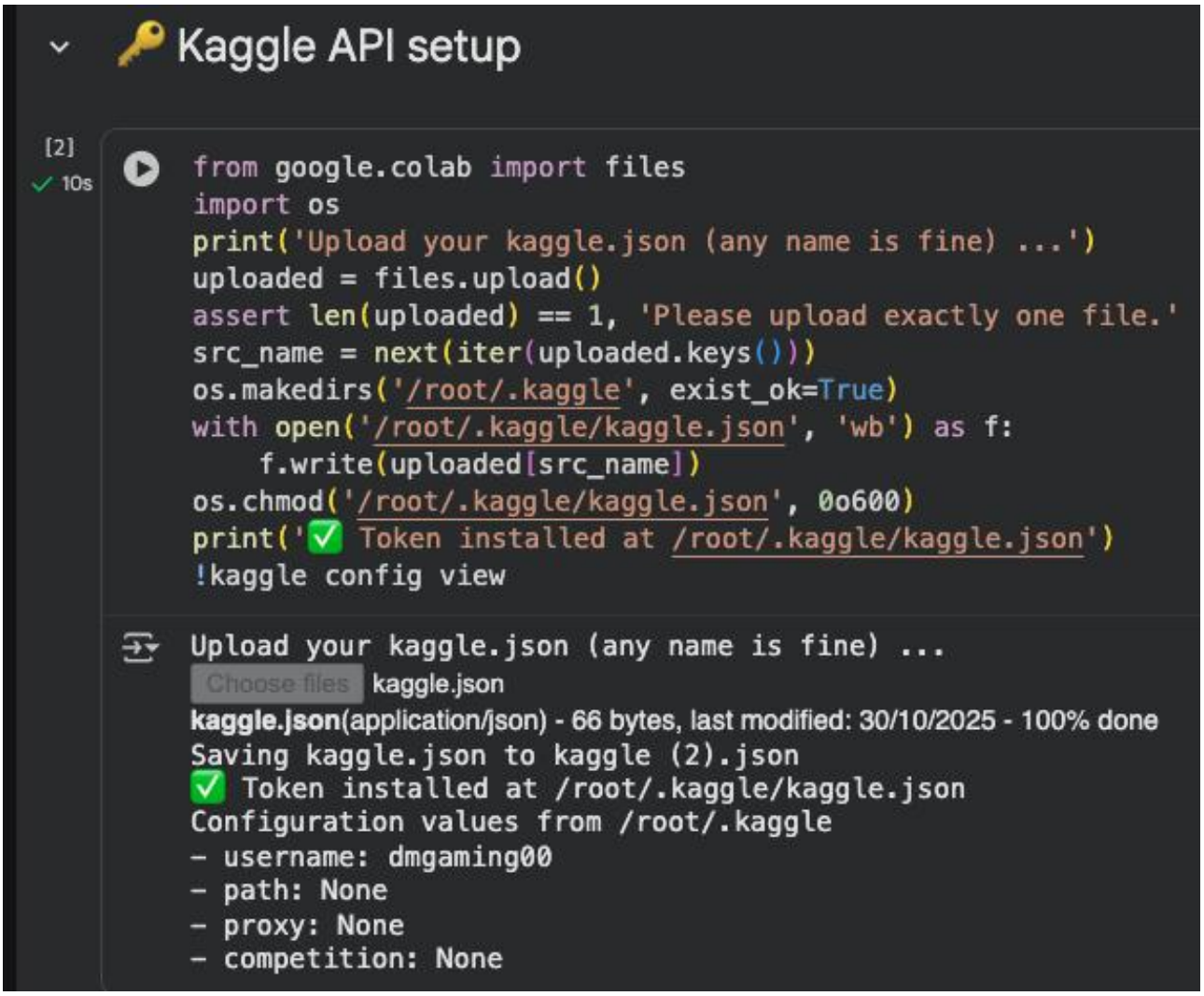
❖ **Submitted By** :- Dev Mulchandani

❖ Overview:-

In the IEEE Fraud Detection AutoGluon Colab, I built a complete machine learning workflow to predict fraudulent transactions using AutoGluon's tabular model. I began by installing AutoGluon and connecting my Kaggle account to automatically download the competition dataset. Since the dataset was large, I optimized the notebook for Colab's limited RAM by dropping long text columns and sampling part of the data. I then trained a lightweight LightGBM model with AutoGluon's TabularPredictor, which handled preprocessing, encoding, and model tuning automatically. After training, I generated probability-based predictions for the test data in smaller chunks to prevent crashes, and saved the output as my_submission.csv for Kaggle submission. This Colab demonstrated how AutoGluon can manage complex, large-scale tabular data efficiently while simplifying model building and deployment through automation.

❖ Screenshots :-





⚙️ Config

[3] ✓ 0s

```
COMPETITION = 'ieee-fraud-detection'
LABEL = 'isFraud'
TIME_LIMIT = 600
MAX_ROWS = 200_000
print('CONFIG OK')
```

```
CONFIG OK
```

✅ Check access & list files

[4] ✓ 1s

```
import subprocess, textwrap
res = subprocess.run(['kaggle', 'competitions', 'files', '-c', COMPETITION], capture_output=True, text=True)
if res.returncode != 0:
    print(res.stderr)
    raise SystemExit(textwrap.dedent('''\
⚠️ Kaggle access issue.
• Join the competition and accept rules on the website.
• Then re-run this cell.
'''))
print(res.stdout)
```

```
name                        size  creationDate
--------------------  ----------  -------------------------
sample_submission.csv     6080314  2019-07-15 00:19:01.536000
test_identity.csv        25797161  2019-07-15 00:19:01.536000
test_transaction.csv    613194934  2019-07-15 00:19:01.536000
train_identity.csv       26529680  2019-07-15 00:19:01.536000
train_transaction.csv   683351067  2019-07-15 00:19:01.536000
```

⬇️ Download & extract data

[5] ✓ 16s

```
import glob, zipfile, os

print("Downloading from Kaggle...")
!kaggle competitions download -c ieee-fraud-detection -p /content/data

print("Extracting ZIP files...")
for z in glob.glob('/content/data/*.zip'):
    with zipfile.ZipFile(z, 'r') as f:
        f.extractall('/content/data')

print("✅ Data ready in /content/data")
!ls -lh /content/data
```

```
Downloading from Kaggle...
ieee-fraud-detection.zip: Skipping, found more recently modified local copy (use --force to force download)
Extracting ZIP files...
✅ Data ready in /content/data
total 1.4G
-rw-r--r-- 1 root root 119M Dec 11  2019 ieee-fraud-detection.zip
-rw-r--r-- 1 root root 5.8M Nov  2 02:03 sample_submission.csv
-rw-r--r-- 1 root root  25M Nov  2 02:03 test_identity.csv
-rw-r--r-- 1 root root 585M Nov  2 02:03 test_transaction.csv
-rw-r--r-- 1 root root  26M Nov  2 02:03 train_identity.csv
-rw-r--r-- 1 root root 652M Nov  2 02:03 train_transaction.csv
```

[6] ✓ 0s

```python
from pathlib import Path

# Define base and data/model paths
BASE = Path("/content")          # or your Google Drive path if using USE_DRIVE=True
DATA_DIR = BASE / "data"
MODEL_DIR = BASE / "AutoGluonModels"

# Make sure folders exist
DATA_DIR.mkdir(parents=True, exist_ok=True)
MODEL_DIR.mkdir(parents=True, exist_ok=True)

# Optional training limit and row cap for low-RAM
TIME_LIMIT = 600
MAX_ROWS = 200_000

print("CONFIG OK")
print("DATA_DIR =", DATA_DIR)
```

```
CONFIG OK
DATA_DIR = /content/data
```

RAM-friendly preprocessing

[7] ✓ 40s

```python
import pandas as pd, gc
train_tr = pd.read_csv(DATA_DIR / 'train_transaction.csv')
train_id = pd.read_csv(DATA_DIR / 'train_identity.csv')
train = train_tr.merge(train_id, on='TransactionID', how='left')
del train_tr, train_id; gc.collect()
print('Raw train:', train.shape)
long_text = [c for c in train.columns if train[c].dtype=='object' and train[c].astype(str).str.len().mean() > 30]
DROP = set(long_text + ['TransactionID'])
X = train.drop(columns=[c for c in DROP if c in train.columns])
print('After drop:', X.shape, '| dropped:', len(DROP))
if len(X) > MAX_ROWS:
    X = X.sample(MAX_ROWS, random_state=0)
print('After sample:', X.shape)
```

```
Raw train: (590540, 434)
After drop: (590540, 433) | dropped: 1
After sample: (200000, 433)
```

🧠 Train (LightGBM only)

[8] ✓ 1m

```python
from autogluon.tabular import TabularPredictor

# Keep only valid model configs (remove None values)
hp = {
    'GBM': [{'num_boost_round': 200}],   # LightGBM only
    # You can add others like 'RF': [{}], 'XT': [{}] later if you want
}

predictor = TabularPredictor(
    label=LABEL,
    eval_metric='roc_auc',
    path=str(MODEL_DIR)
).fit(
    X,
    hyperparameters=hp,
    presets=None,
    num_bag_folds=0,
    num_stack_levels=0,
    time_limit=TIME_LIMIT,
    verbosity=2
)

predictor.fit_summary()
```

```
Warning: path already exists! This predictor may overwrite an existing predictor! path="/content/AutoGluonModels"
Verbosity: 2 (Standard Logging)
=================== System Info ===================
AutoGluon Version:  1.4.0
Python Version:     3.12.12
Operating System:   Linux
Platform Machine:   x86_64
Platform Version:   #1 SMP Thu Oct  2 10:42:05 UTC 2025
CPU Count:          2
Memory Avail:       7.38 GB / 12.67 GB (58.2%)
Disk Space Avail:   60.14 GB / 107.72 GB (55.8%)
===================================================
No presets specified! To achieve strong results with AutoGluon, it is recommended to use the available presets. D
        Recommended Presets (For more details refer to https://auto.gluon.ai/stable/tutorials/tabular/tabular-ess
        presets='extreme' : New in v1.4: Massively better than 'best' on datasets <30000 samples by using new mod
        presets='best'    : Maximize accuracy. Recommended for most users. Use in competitions and benchmarks.
        presets='high'    : Strong accuracy with fast inference speed.
        presets='good'    : Good accuracy with very fast inference speed.
        presets='medium'  : Fast training time, ideal for initial prototyping.
Beginning AutoGluon training ... Time limit = 600s
AutoGluon will save models to "/content/AutoGluonModels"
Train Data Rows:    200000
Train Data Columns: 432
Label Column:       isFraud
AutoGluon infers your prediction problem is: 'binary' (because only two unique label-values observed).
        2 unique label values:  [np.int64(0), np.int64(1)]
        If 'binary' is not the correct problem_type, please manually specify the problem_type parameter during Pr
Problem Type:       binary
Preprocessing data ...
Selected class <--> label mapping:  class 1 = 1, class 0 = 0
Using Feature Generators to preprocess the data ...
Fitting AutoMLPipelineFeatureGenerator...
        Available Memory:                    7787.25 MB
        Train Data (Original)  Memory Usage: 848.28 MB (10.9% of available memory)
        Warning: Data size prior to feature transformation consumes 10.9% of available memory. Consider increasin
        Inferring data type of each feature based on column values. Set feature_metadata_in to manually specify s
        Stage 1 Generators:
                Fitting AsTypeFeatureGenerator...
        Stage 2 Generators:
                Fitting FillNaFeatureGenerator...
        Stage 3 Generators:
                Fitting IdentityFeatureGenerator...
                Fitting CategoryFeatureGenerator...
                        Fitting CategoryMemoryMinimizeFeatureGenerator...
```

```
 'model_fit_times': {'LightGBM': 51.43282175064087,
  'WeightedEnsemble_L2': 0.0052874088287353516},
 'model_pred_times': {'LightGBM': 0.09017086029052734,
  'WeightedEnsemble_L2': 0.0008881092071533203},
 'num_bag_folds': 0,
 'max_stack_level': 2,
 'num_classes': 2,
 'model_hyperparams': {'LightGBM': {'learning_rate': 0.05,
   'num_boost_round': 200},
  'WeightedEnsemble_L2': {'use_orig_features': False,
   'valid_stacker': True,
   'max_base_models': 0,
   'max_base_models_per_type': 'auto',
   'save_bag_folds': True,
   'stratify': 'auto',
   'bin': 'auto',
   'n_bins': None}},
 'leaderboard':                 model  score_val eval_metric  pred_time_val   fit_time  \
0             LightGBM   0.929648     roc_auc       0.090171  51.432822
1  WeightedEnsemble_L2   0.929648     roc_auc       0.091059  51.438109

   pred_time_val_marginal  fit_time_marginal  stack_level  can_infer  \
0                0.090171          51.432822            1       True
1                0.000888           0.005287            2       True

   fit_order
0          1
1          2  }
```

🔮 Predict & create submission

[13] ✓ 57s

```
import pandas as pd, gc, os
key = 'TransactionID'

# 1) Raw training features (AutoGluon ≥1.4)
needed_feats = list(predictor.feature_metadata.get_features())
print("Total features expected by model:", len(needed_feats))

# 2) Map columns to files
tx_head = pd.read_csv(DATA_DIR / 'test_transaction.csv', nrows=5)
id_head = pd.read_csv(DATA_DIR / 'test_identity.csv', nrows=5)
tx_cols_all, id_cols_all = set(tx_head.columns), set(id_head.columns)

tx_usecols = sorted((set(needed_feats) & tx_cols_all) | {key})
id_usecols = sorted((set(needed_feats) & id_cols_all) | {key})

# 3) Load identity once (smaller)
id_small = pd.read_csv(DATA_DIR / 'test_identity.csv', usecols=id_usecols).set_index(key)
print("Identity small shape:", id_small.shape)

# 4) Prepare submission
sub_path = DATA_DIR / 'my_submission.csv'
with open(sub_path, 'w') as f:
    f.write(f"{key},{LABEL}\n")

def to_positive_class_proba(p):
```

[13] ✓ 57s

```
def to_positive_class_proba(p):
    """Return a 1-D numpy array of positive-class probabilities, regardless of shape/type."""
    import numpy as np
    if isinstance(p, pd.DataFrame):
        # Column names are class labels
        pos = getattr(predictor, "positive_class", 1)
        if pos in p.columns:
            return p[pos].to_numpy()
        else:
            return p.iloc[:, -1].to_numpy()   # fallback: last col
    if isinstance(p, pd.Series):
        return p.to_numpy()
    arr = np.asarray(p)
    if arr.ndim == 2:
        return arr[:, -1]                     # use last column as positive class
    return arr

# 5) Predict in chunks (adjust if RAM tight)
chunksize = 100_000
total_rows = 0

for i, chunk in enumerate(pd.read_csv(DATA_DIR / 'test_transaction.csv',
                                      usecols=tx_usecols,
                                      chunksize=chunksize), 1):
    # Join identity features
    chunk = chunk.set_index(key).join(id_small, how='left').reset_index()

    # Keep only model features
    keep = [c for c in needed_feats if c in chunk.columns]
    X_chunk = chunk[keep]

    # If any expected features are missing, add them as NaN (AG can handle)
    missing = [c for c in needed_feats if c not in X_chunk.columns]
    for c in missing:
        X_chunk[c] = float('nan')

    # Ensure column order matches (not strictly required, but tidy)
    X_chunk = X_chunk[[c for c in needed_feats]]

    # Predict and always convert to 1-D positive-class proba
    proba = to_positive_class_proba(predictor.predict_proba(X_chunk))
```

```
        out = pd.DataFrame({key: chunk[key].values, LABEL: proba})
        out.to_csv(sub_path, mode='a', header=False, index=False)

        total_rows += len(out)
        print(f"Chunk {i}: wrote {len(out):,} rows (total {total_rows:,})")
        del chunk, X_chunk, proba, out
        gc.collect()

    print("✅ Saved submission to:", sub_path)
    !ls -lh "$sub_path"
    !head -n 5 "$sub_path"
```

```
Total features expected by model: 427
Identity small shape: (141907, 2)
Chunk 1: wrote 100,000 rows (total 100,000)
Chunk 2: wrote 100,000 rows (total 200,000)
Chunk 3: wrote 100,000 rows (total 300,000)
Chunk 4: wrote 100,000 rows (total 400,000)
Chunk 5: wrote 100,000 rows (total 500,000)
Chunk 6: wrote 6,691 rows (total 506,691)
✅ Saved submission to: /content/data/my_submission.csv
-rw-r--r-- 1 root root 14M Nov  2 02:13 /content/data/my_submission.csv
TransactionID,isFraud
3663549,0.004711710382252932
3663550,0.008591421879827976
3663551,0.019612561911344528
3663552,0.0031595013570040464
```