AAM-IPL-Wk-4-LogiReg-Cross-Sell-Vehicle-Insurance-Prediction-Full-Code-V3

October 5, 2024

1 AAM-IPL Week-3 Logistic Regression - Cross-Sell Vehicle Insurance Prediction Reference Implementation

Implemented By: Venkateswar Reddy Melachervu **Branch of Study**: **Department**: **Semester**: **Email**: **Guest Faculty and Instructor**: Venkateswar Reddy Melachervu, CTO, Brillium Technologies **Program Coordinator**: Prof. V.Suresh, CSM, GPREC

Project Implementation Details: As published in the project announcement in AAM-IPL Online Classroom

AAM-IPL of GPREC is brought to you by Brillium Technologies.

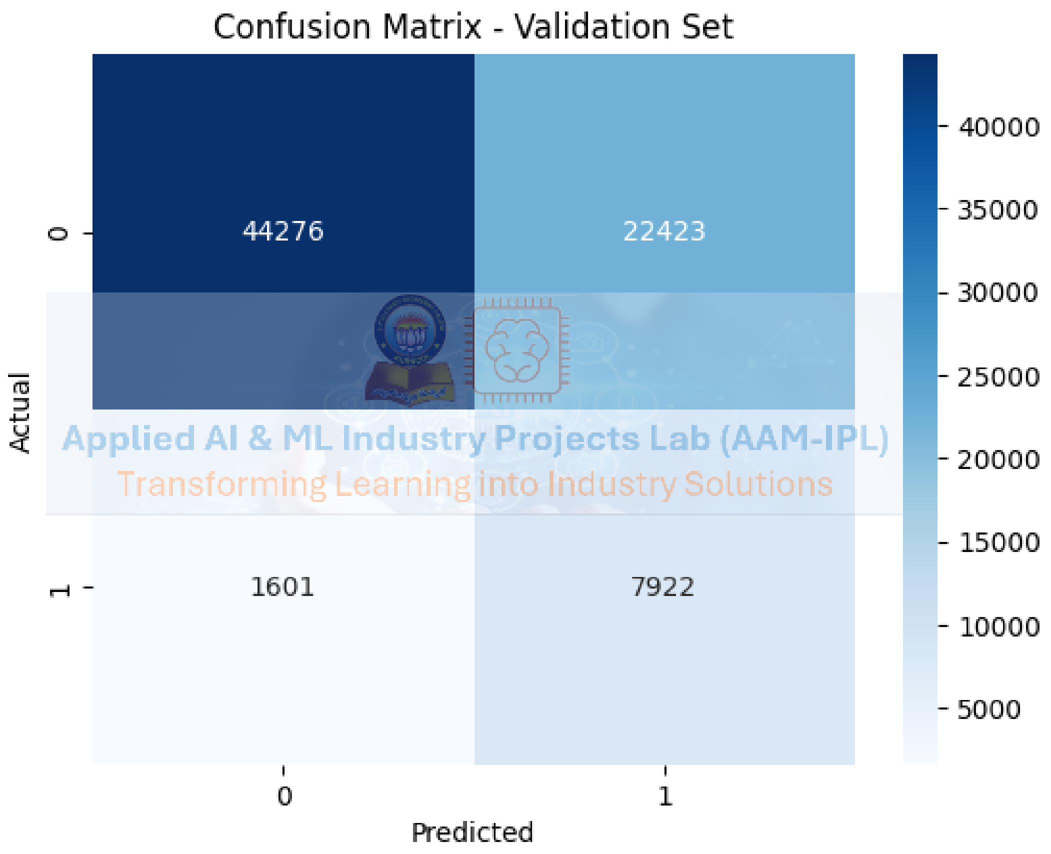
[2]:
```
# Cross-Sell Vehicle Insurance Prediction with Watermark

# Importing the necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV,␣
 ↪cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report, roc_curve, roc_auc_score, precision_recall_curve
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.offsetbox import TextArea, DrawingArea, OffsetImage,␣
 ↪AnnotationBbox
import matplotlib.image as mpimg

# Load the watermark image
watermark = mpimg.imread('AAM-IPL-Header-6.png')

# Function to add watermark
def add_watermark(ax, watermark, alpha=0.3, zoom=0.30):
    imagebox = OffsetImage(watermark, alpha=alpha, zoom=zoom)
    ab = AnnotationBbox(imagebox, (0.5, 0.5), frameon=False, xycoords='axes␣
 ↪fraction', boxcoords="axes fraction", pad=0)
    ax.add_artist(ab)

# Load the training and test datasets
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Check for missing values
print("Missing values in training data:\n", train_data.isnull().sum())
print("\nMissing values in test data:\n", test_data.isnull().sum())

# Data Preprocessing
label_encoders = {}
from sklearn.preprocessing import LabelEncoder

# Encoding 'Gender'
label_encoders['Gender'] = LabelEncoder()
train_data['Gender'] = label_encoders['Gender'].
 ↪fit_transform(train_data['Gender'])
test_data['Gender'] = label_encoders['Gender'].transform(test_data['Gender'])

# Encoding 'Vehicle_Age'
label_encoders['Vehicle_Age'] = LabelEncoder()
train_data['Vehicle_Age'] = label_encoders['Vehicle_Age'].
 ↪fit_transform(train_data['Vehicle_Age'])
test_data['Vehicle_Age'] = label_encoders['Vehicle_Age'].
 ↪transform(test_data['Vehicle_Age'])

# Encoding 'Vehicle_Damage'
label_encoders['Vehicle_Damage'] = LabelEncoder()
train_data['Vehicle_Damage'] = label_encoders['Vehicle_Damage'].
 ↪fit_transform(train_data['Vehicle_Damage'])
test_data['Vehicle_Damage'] = label_encoders['Vehicle_Damage'].
 ↪transform(test_data['Vehicle_Damage'])

# Separate features (X) and target/response variable (Y) from the training data
X = train_data.drop(columns=['Response', 'id'])
y = train_data['Response']

# Splitting the training data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Handling class imbalance using SMOTE
smote = SMOTE(random_state=42)
```

```
X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)

# Scaling the data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train_smote)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(test_data.drop(columns=['id', 'Response']))

# Hyperparameter tuning using GridSearchCV
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'saga']
}

logreg = LogisticRegression(class_weight='balanced', max_iter=10000)
grid_search = GridSearchCV(logreg, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train_scaled, y_train_smote)

# Best hyperparameters
print("Best Parameters from Grid Search:", grid_search.best_params_)

# Model Evaluation on Validation Set
best_logreg = grid_search.best_estimator_
y_val_pred = best_logreg.predict(X_val_scaled)

# Accuracy Score on validation set
accuracy_val = accuracy_score(y_val, y_val_pred)
print(f"Validation Accuracy: {accuracy_val * 100:.4f}%")

# Confusion matrix on validation set
conf_matrix_val = confusion_matrix(y_val, y_val_pred)
fig, ax = plt.subplots()
sns.heatmap(conf_matrix_val, annot=True, fmt='d', cmap='Blues', ax=ax)
plt.title('Confusion Matrix - Validation Set')
plt.xlabel('Predicted')
plt.ylabel('Actual')
add_watermark(ax, watermark)
plt.show()

# Classification report
class_report_val = classification_report(y_val, y_val_pred)
print("Classification Report (Validation Set):\n", class_report_val)

# ROC Curve
y_val_prob = best_logreg.predict_proba(X_val_scaled)[:,1]
fpr, tpr, _ = roc_curve(y_val, y_val_prob)
roc_auc = roc_auc_score(y_val, y_val_prob)
```

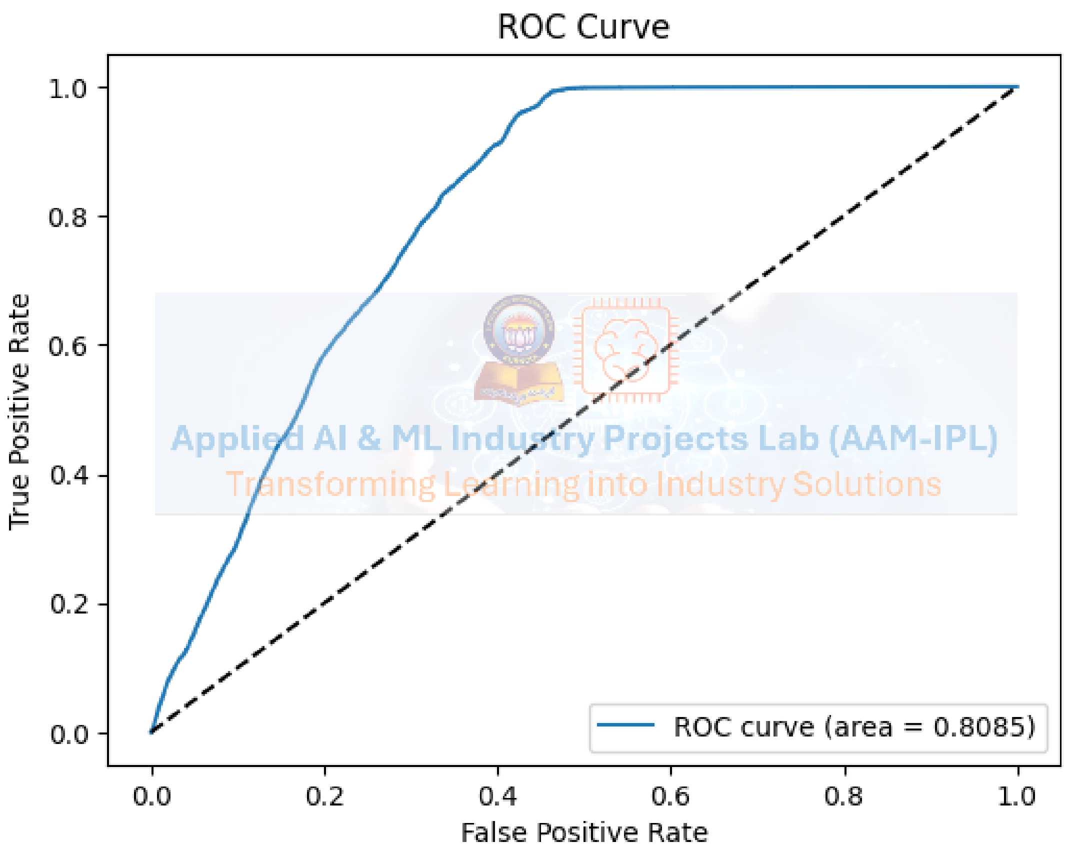
```
fig, ax = plt.subplots()
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
add_watermark(ax, watermark)
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(y_val, y_val_prob)
fig, ax = plt.subplots()
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
add_watermark(ax, watermark)
plt.show()

# Feature Importance using coefficients from logistic regression
importance = best_logreg.coef_[0]
feature_names = X.columns
feature_importances = pd.Series(importance, index=feature_names).
 ↪sort_values(ascending=False)

fig, ax = plt.subplots(figsize=(10,6))
sns.barplot(x=feature_importances, y=feature_importances.index, ax=ax)
plt.title('Feature Importance - Logistic Regression')
plt.xlabel('Coefficient Value')
plt.ylabel('Feature')
add_watermark(ax, watermark)
plt.show()

# Predicting on the test dataset
test_predictions_balanced = best_logreg.predict(X_test_scaled)

# Prepare the submission file for test predictions
submission_balanced = pd.DataFrame({
    'id': test_data['id'],
    'Response': test_predictions_balanced
})

# Save the submission file
submission_balanced.to_csv('insurance_prediction_submission_balanced.csv',␣
 ↪index=False)
```

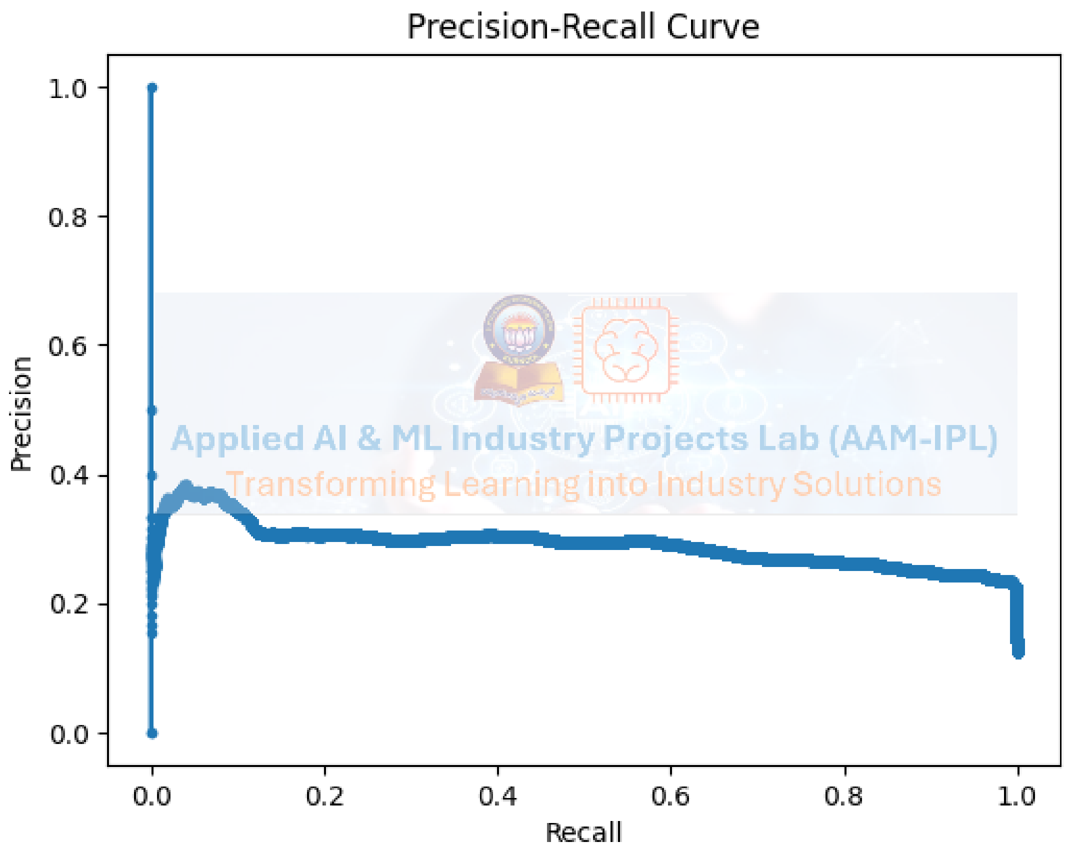
```python
print("Submission file created: 'insurance_prediction_submission_balanced.csv'")
```

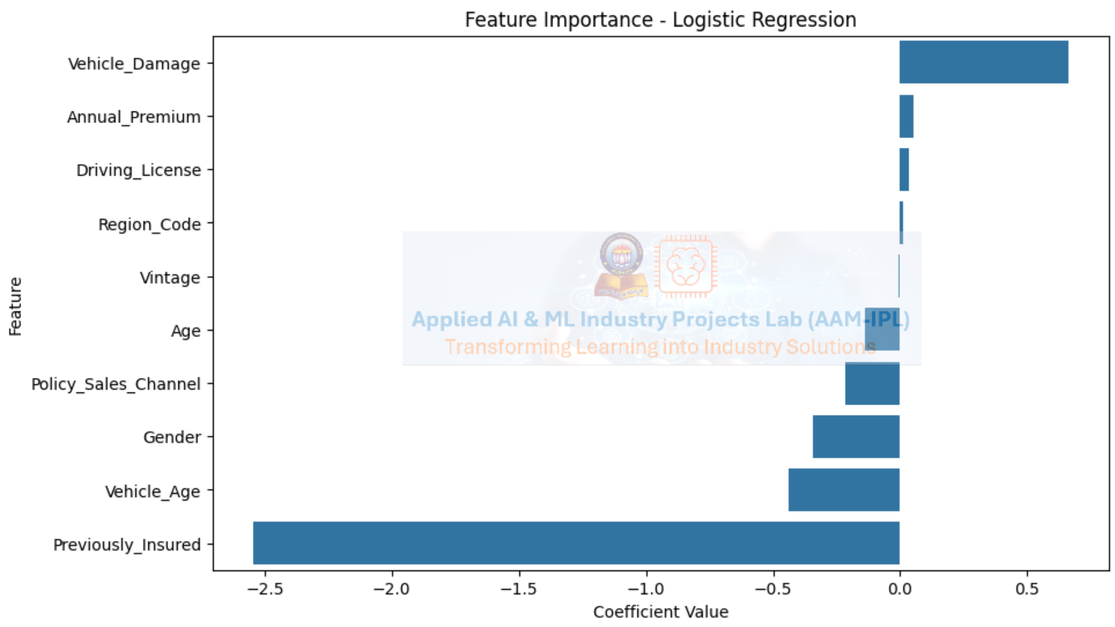
```
Missing values in training data:
 id                      0
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
Response                0
dtype: int64

Missing values in test data:
 id                           0
Gender                       0
Age                          0
Driving_License              0
Region_Code                  0
Previously_Insured           0
Vehicle_Age                  0
Vehicle_Damage               0
Annual_Premium               0
Policy_Sales_Channel         0
Vintage                      0
Response                127037
dtype: int64
Best Parameters from Grid Search: {'C': 0.01, 'solver': 'saga'}
Validation Accuracy: 68.4815%
```



```
Classification Report (Validation Set):
               precision    recall  f1-score   support

           0       0.97      0.66      0.79     66699
           1       0.26      0.83      0.40      9523

    accuracy                           0.68     76222
   macro avg       0.61      0.75      0.59     76222
weighted avg       0.88      0.68      0.74     76222
```


ROC Curve
True Positive Rate
False Positive Rate
ROC curve (area = 0.8085)
0.0
0.2
0.4
0.6
0.8
1.0


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Feature Importance - Logistic Regression
Feature
Coefficient Value
Vehicle_Damage
Annual_Premium
Driving_License
Region_Code
Vintage
Age
Policy_Sales_Channel
Gender
Vehicle_Age
Previously_Insured
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5

```
Submission file created: 'insurance_prediction_submission_balanced.csv'
```

[3]:
```
# Generate the PDF of code and output of project jupyter file
!jupyter nbconvert --to pdf␣
 ↪AAM-IPL-Wk-4-LogiReg-Cross-Sell-Vehicle-Insurance-Prediction-Full-Code-V3.
 ↪ipynb
```

```
This application is used to convert notebook files (*.ipynb)
        to various other formats.

        WARNING: THE COMMANDLINE INTERFACE MAY CHANGE IN FUTURE RELEASES.

Options
=======
The options below are convenience aliases to configurable class-options,
as listed in the "Equivalent to" description-line of the aliases.
To see all configurable class-options for some <cmd>, use:
    <cmd> --help-all

--debug
    set log level to logging.DEBUG (maximize logging output)
    Equivalent to: [--Application.log_level=10]
--show-config
    Show the application's configuration (human-readable format)
    Equivalent to: [--Application.show_config=True]
--show-config-json
    Show the application's configuration (json format)
    Equivalent to: [--Application.show_config_json=True]
--generate-config
    generate default config file
    Equivalent to: [--JupyterApp.generate_config=True]
-y
    Answer yes to any questions instead of prompting.
    Equivalent to: [--JupyterApp.answer_yes=True]
--execute
    Execute the notebook prior to export.
    Equivalent to: [--ExecutePreprocessor.enabled=True]
--allow-errors
    Continue notebook execution even if one of the cells throws an error and
include the error message in the cell output (the default behaviour is to abort
conversion). This flag is only relevant if '--execute' was specified, too.
    Equivalent to: [--ExecutePreprocessor.allow_errors=True]
--stdin
    read a single notebook file from stdin. Write the resulting notebook with
default basename 'notebook.*'
    Equivalent to: [--NbConvertApp.from_stdin=True]
--stdout
    Write notebook output to stdout instead of files.
    Equivalent to: [--NbConvertApp.writer_class=StdoutWriter]
--inplace
    Run nbconvert in place, overwriting the existing notebook (only
            relevant when converting to notebook format)
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=]
--clear-output
    Clear output of current file and save in place,
            overwriting the existing notebook.
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--ClearOutputPreprocessor.enabled=True]
--coalesce-streams
    Coalesce consecutive stdout and stderr outputs into one stream (within each
cell).
    Equivalent to: [--NbConvertApp.use_output_suffix=False
--NbConvertApp.export_format=notebook --FilesWriter.build_directory=
--CoalesceStreamsPreprocessor.enabled=True]
--no-prompt
    Exclude input and output prompts from converted document.
    Equivalent to: [--TemplateExporter.exclude_input_prompt=True
--TemplateExporter.exclude_output_prompt=True]
--no-input
    Exclude input cells and output prompts from converted document.
            This mode is ideal for generating code-free reports.
    Equivalent to: [--TemplateExporter.exclude_output_prompt=True
--TemplateExporter.exclude_input=True
--TemplateExporter.exclude_input_prompt=True]
--allow-chromium-download
    Whether to allow downloading chromium if no suitable version is found on the
system.
    Equivalent to: [--WebPDFExporter.allow_chromium_download=True]
--disable-chromium-sandbox
    Disable chromium security sandbox when converting to PDF..
    Equivalent to: [--WebPDFExporter.disable_sandbox=True]
--show-input
    Shows code input. This flag is only useful for dejavu users.
    Equivalent to: [--TemplateExporter.exclude_input=False]
--embed-images
    Embed the images as base64 dataurls in the output. This flag is only useful
for the HTML/WebPDF/Slides exports.
    Equivalent to: [--HTMLExporter.embed_images=True]
--sanitize-html
    Whether the HTML in Markdown cells and cell outputs should be sanitized..
    Equivalent to: [--HTMLExporter.sanitize_html=True]
--log-level=<Enum>
    Set the log level by value or name.
    Choices: any of [0, 10, 20, 30, 40, 50, 'DEBUG', 'INFO', 'WARN', 'ERROR',
'CRITICAL']
    Default: 30
    Equivalent to: [--Application.log_level]
--config=<Unicode>
    Full path of a config file.
    Default: ''
    Equivalent to: [--JupyterApp.config_file]
--to=<Unicode>
    The export format to be used, either one of the built-in formats
            ['asciidoc', 'custom', 'html', 'latex', 'markdown', 'notebook',
'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides', 'webpdf']
            or a dotted object name that represents the import path for an
            ``Exporter`` class
    Default: ''
    Equivalent to: [--NbConvertApp.export_format]
--template=<Unicode>
    Name of the template to use
    Default: ''
    Equivalent to: [--TemplateExporter.template_name]
--template-file=<Unicode>
    Name of the template file to use
    Default: None
    Equivalent to: [--TemplateExporter.template_file]
--theme=<Unicode>
    Template specific theme(e.g. the name of a JupyterLab CSS theme distributed
    as prebuilt extension for the lab template)
    Default: 'light'
    Equivalent to: [--HTMLExporter.theme]
--sanitize_html=<Bool>
    Whether the HTML in Markdown cells and cell outputs should be sanitized.This
    should be set to True by nbviewer or similar tools.
    Default: False
    Equivalent to: [--HTMLExporter.sanitize_html]
--writer=<DottedObjectName>
    Writer class used to write the
                                        results of the conversion
    Default: 'FilesWriter'
    Equivalent to: [--NbConvertApp.writer_class]
--post=<DottedOrNone>
    PostProcessor class used to write the
                                        results of the conversion
    Default: ''
    Equivalent to: [--NbConvertApp.postprocessor_class]
--output=<Unicode>
    Overwrite base name use for output files.
                Supports pattern replacements '{notebook_name}'.
    Default: '{notebook_name}'
    Equivalent to: [--NbConvertApp.output_base]
```

```
--output-dir=<Unicode>
    Directory to write output(s) to. Defaults
                                  to output to the directory of each notebook.
To recover
                                  previous default behaviour (outputting to the
current
                                  working directory) use . as the flag value.
    Default: ''
    Equivalent to: [--FilesWriter.build_directory]
--reveal-prefix=<Unicode>
    The URL prefix for reveal.js (version 3.x).
            This defaults to the reveal CDN, but can be any url pointing to a
copy
            of reveal.js.
            For speaker notes to work, this must be a relative path to a local
            copy of reveal.js: e.g., "reveal.js".
            If a relative path is given, it must be a subdirectory of the
            current directory (from which the server is run).
            See the usage documentation
            (https://nbconvert.readthedocs.io/en/latest/usage.html#reveal-js-
html-slideshow)
            for more details.
    Default: ''
    Equivalent to: [--SlidesExporter.reveal_url_prefix]
--nbformat=<Enum>
    The nbformat version to write.
            Use this to downgrade notebooks.
    Choices: any of [1, 2, 3, 4]
    Default: 4
    Equivalent to: [--NotebookExporter.nbformat_version]

Examples
--------

    The simplest way to use nbconvert is

            > jupyter nbconvert mynotebook.ipynb --to html

            Options include ['asciidoc', 'custom', 'html', 'latex', 'markdown',
'notebook', 'pdf', 'python', 'qtpdf', 'qtpng', 'rst', 'script', 'slides',
'webpdf'].

            > jupyter nbconvert --to latex mynotebook.ipynb

            Both HTML and LaTeX support multiple output templates. LaTeX
includes
            'base', 'article' and 'report'.  HTML includes 'basic', 'lab' and
            'classic'. You can specify the flavor of the format used.
            > jupyter nbconvert --to html --template lab mynotebook.ipynb

            You can also pipe the output to stdout, rather than a file

            > jupyter nbconvert mynotebook.ipynb --stdout

            PDF is generated via latex

            > jupyter nbconvert mynotebook.ipynb --to pdf

            You can get (and serve) a Reveal.js-powered slideshow

            > jupyter nbconvert myslides.ipynb --to slides --post serve

            Multiple notebooks can be given at the command line in a couple of
            different ways:

            > jupyter nbconvert notebook*.ipynb
            > jupyter nbconvert notebook1.ipynb notebook2.ipynb

            or you can specify the notebooks list in a config file, containing::

                c.NbConvertApp.notebooks = ["my_notebook.ipynb"]

            > jupyter nbconvert --config mycfg.py

To see all available configurables, use `--help-all`.

[NbConvertApp] WARNING | pattern 'AAM-IPL-Wk-4-LogiReg-Cross-Sell-Vehicle-
Insurance-Prediction-Full-Code.ipynb' matched no files
```