EXAM_EBA3630

May 8, 2025

1 Take-home exam in EBA 3630 Data Driven Management Accounting

1.0.1 Declaration of AI Use

AI tools that have been used in the work on assignment/exam:

- **Name (and version) of the AI tool:** ChatGPT 4

Purpose of AI use:

I have used AI while undertaking my assignment in the following ways:

☐ To generate ideas

☐ To explain concepts

☐ To support my use of language

☐ To organize data

☐ To analyze data

☐ To visualize data

☒ To debug code

☐ To write code

☒ Other (please specify): Grammar Checker

Please indicate the extent to which you have utilized AI in completing this exam: 30%

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.patches as mpatches
from matplotlib import pyplot as plt

import plotly.express as px
import ipywidgets
```

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.metrics import accuracy_score, confusion_matrix, make_scorer

import openpyxl
```

[2]:
```
#pip install plotly ipywidgets openpyxl
```

2 Exercise 1

2.1 Q1.1) Develop an interactive visualization of the Balanced Scorecard that enables users to select specific markets for display. Include performance markers to represent the achievement level of the targets using a color-coded system: green for targets that are met or exceeded, orange for targets missed by no more than 10%, and red for targets that fall short.

[3]:
```
df_ex1 = pd.read_excel("Exercise1.xlsx")

df_ex1["Deviation %"] = (df_ex1["Actual Performance"] - df_ex1["Target␣
 ↪Performance"]) / df_ex1["Target Performance"]

def classify_target(row):
    actual = row["Actual Performance"]
    target = row["Target Performance"]
    deviation = (actual - target) / target

    if row["Perspective"] == "Internal Business Process Perspective":
        if actual <= target:
            return "yes"
        elif deviation <= 0.10:
            return "close"
        else:
            return "no"
    else:
        if actual >= target:
            return "yes"
        elif deviation >= -0.10:
            return "close"
        else:
            return "no"

df_ex1["Target Reached"] = df_ex1.apply(classify_target, axis=1)

def reached_target(row):
    green = "background-color: green;"
```

```
    orange = "background-color: orange;"
    red = "background-color: red;"
    default = ""

    if row["Target Reached"] == "yes":
        return [default, green]
    elif row["Target Reached"] == "close":
        return [default, orange]
    else:
        return [default, red]

drop_down = ipywidgets.Dropdown(
    options=df_ex1["Market"].unique(),
    value=df_ex1["Market"].unique()[0],
    description="Market:",
    style={"description_width": "initial"}
)

def scorecard(Market):
    df_ex1_filter = df_ex1[df_ex1["Market"] == Market]
    df_ex1_filter = df_ex1_filter[[
        "Perspective", "Measure", "Target Performance", "Actual Performance",␣
 ↪"Target Reached","Deviation %"
    ]]
    return df_ex1_filter.style.apply(reached_target, subset=["Actual␣
 ↪Performance", "Target Reached"], axis=1)

ipywidgets.interact(scorecard, Market=drop_down);
```

```
interactive(children=(Dropdown(description='Market:', options=('North America',␣
 ↪'South America', 'Europe'), st…
```

Note: The Internal Business Process Perspective is optimized for **lower is better**, meaning that a higher Actual Performance than Target Performance indicates underperformance. For all other perspectives, **higher is better**.

2.2 Q1.2) Select three measures from GEAR's Balanced Scorecard and analyze how they contribute (or fail to contribute) to the implementation of GEAR's strategy and value generation.

1. Product Defect Rate - North America: Target: 0.020, Actual: 0.030 = Not Reached
- South America: Target: 0.050, Actual: 0.060 = Not Reached
- Europe: Target: 0.020, Actual: 0.030 = Not Reached

This measure directly affects customer satisfaction and retention. Missing the target across all regions suggests potential quality control issues that could damage GEAR's brand reputation, reduce long-term customer loyalty and performance in the Customer Perspective.

2. Customer Satisfaction Score - North America: Target: 4.7, Actual: 4.8 = Target Reached
- South America: Target: 4.7, Actual: 4.5 = Close to Target
- Europe: Target: 4.7, Actual: 4.8 = Target Reached

Customer satisfaction is strong in all regions, with only a slight shortfall in South America. This indicates that GEAR is delivering on its customer value proposition and maintaining a positive brand perception globally—supporting customer loyalty and market competitiveness.

3. Customization Upcharge Revenue - North America: Target: 1,500,000, Actual: 1,600,000 = Target Reached
- South America: Target: 900,000, Actual: 800,000 = Not Reached
- Europe: Target: 1,400,000, Actual: 1,400,000 = Target Reached

This measure captures revenue from product customization, which is central to GEAR's differentiation strategy. While most regions hit or exceed the target, underperformance in South America may point to differing customer preferences, suggesting a need to adapt the customization strategy to regional market.

2.3 Q1.3) As GEAR plans to expand into the Asian and Australian markets, the management accounting team plans to integrate these markets into GEAR's Balanced Scorecard framework. They are considering using the same performance measures with minor target adjustments for these new markets. Provide your recommendations on this consideration. (Limit your response to a concise explanation)

Using the same performance measures across markets ensures consistency and comparability, which is valuable for strategic alignment. However, I recommend adjusting targets more than just slightly—based on regional market conditions, customer expectations, labor costs, and operational maturity. Cultural, economic, and logistical differences in Asia and Australia may impact what is realistically achievable and relevant, so localized benchmarking should guide target setting to ensure the Balanced Scorecard remains both fair and motivating.

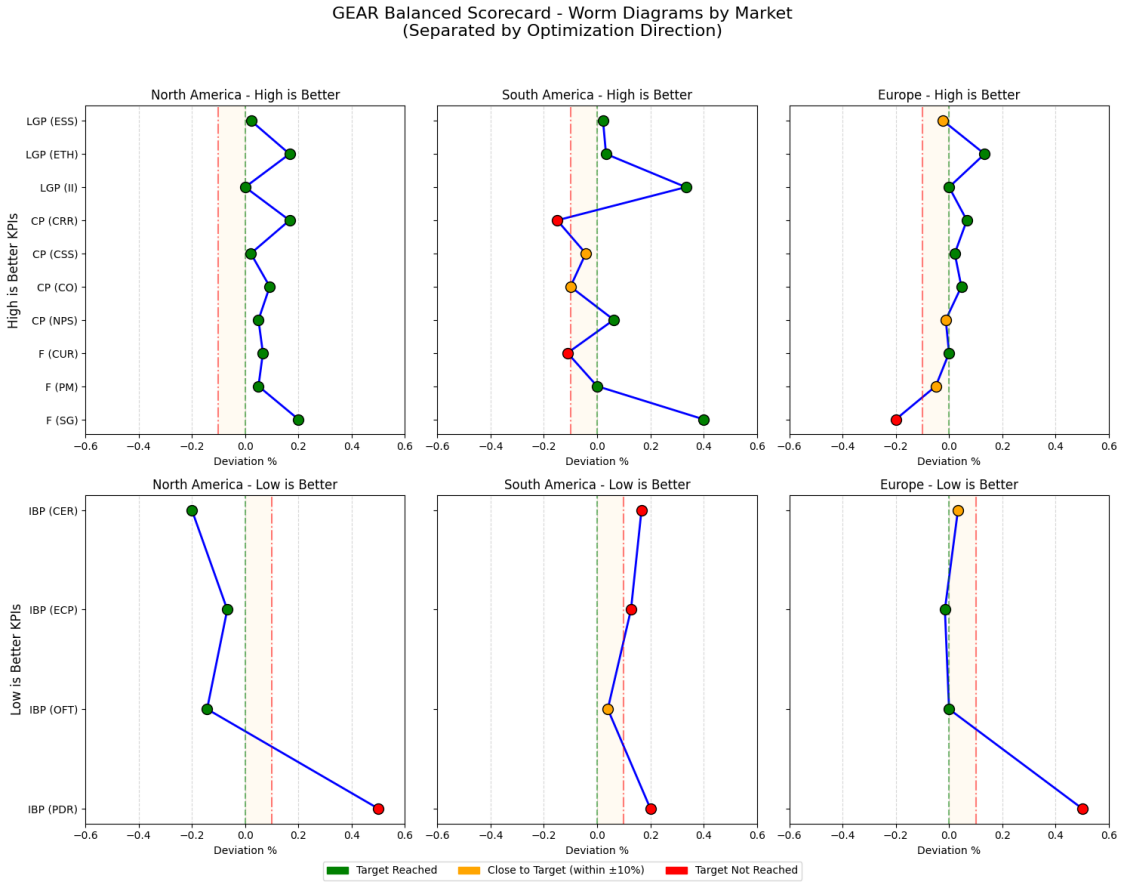
2.4 Q1.4) The management accounting team seeks to enhance the Balanced Scorecard with additional graphical visualizations inspired by a case study example (See below). Design a similar visualization tailored for each of GEAR's markets (instead of contracts).

[4]:
```
conditions = [
    df_ex1["Measure"] == "Employee Training Hours",
    df_ex1["Measure"] == "Innovation Index",
    df_ex1["Measure"] == "Employee Satisfaction Score",
    df_ex1["Measure"] == "Customer Retention Rate",
    df_ex1["Measure"] == "Customization Orders",
    df_ex1["Measure"] == "Customer Satisfaction Score",
    df_ex1["Measure"] == "Net Promoter Score",
    df_ex1["Measure"] == "Customization Upcharge Revenue",
```

```
    df_ex1["Measure"] == "Profit Margin",
    df_ex1["Measure"] == "Sales Growth"
]
choices = ["High"] * len(conditions)
df_ex1["Direction"] = np.select(conditions, choices, default="Low")

df_ex1["Measure Label"] = df_ex1["Perspective"] + "(" + df_ex1["Measure"]

df_ex1["Perspective Abbr"] = df_ex1["Perspective"].map({
    "Learning and Growth Perspective": "LGP",
    "Customer Perspective": "CP",
    "Financial": "F",
    "Internal Business Process Perspective": "IBP"
})

measure_abbr_map = {
    "Employee Training Hours": "ETH",
    "Innovation Index": "II",
    "Employee Satisfaction Score": "ESS",
    "Customer Retention Rate": "CRR",
    "Customization Orders": "CO",
    "Customer Satisfaction Score": "CSS",
    "Net Promoter Score": "NPS",
    "Customization Upcharge Revenue": "CUR",
    "Profit Margin": "PM",
    "Sales Growth": "SG",
    "Product Defect Rate": "PDR",
    "Order Fulfillment Time": "OFT",
    "Efficiency of Customization Process": "ECP",
    "Customization Error Rate": "CER"
}
df_ex1["Measure Abbr"] = df_ex1["Measure"].map(measure_abbr_map)

df_ex1["Compact Label"] = df_ex1["Perspective Abbr"] + " (" + df_ex1["Measure␣
 ↪Abbr"] + ")"

def get_color(status):
    status = status.lower().strip()
    if status == "yes":
        return "green"
    elif status == "no":
        return "red"
    else:
        return "orange"

df_ex1_high = df_ex1[df_ex1["Direction"] == "High"]
df_ex1_low = df_ex1[df_ex1["Direction"] == "Low"]
```

```
regions = df_ex1["Market"].unique()
num_regions = len(regions)

fig, axes = plt.subplots(2, num_regions, figsize=(5 * num_regions, 12),␣
 ↪sharey="row")
if num_regions == 1:
    axes = np.array([[axes[0]], [axes[1]]])

datasets = {"High": df_ex1_high, "Low": df_ex1_low}

# Plot loop
for row_idx, (direction, df) in enumerate(datasets.items()):
    for col_idx, region in enumerate(regions):
        subset = df[df["Market"] == region].copy()
        subset["Perspective Order"] = subset["Perspective"].map({
            "Internal Business Process Perspective": 3,
            "Learning and Growth Perspective": 0,
            "Customer Perspective": 1,
            "Financial": 2
        })
        subset = subset.sort_values(by=["Perspective Order", "Measure"],␣
 ↪ascending=[True, True]).reset_index(drop=True)

        ax = axes[row_idx, col_idx]

        if col_idx == 0:
            ax.yaxis.set_label_position("left")
            ax.yaxis.tick_left()
            ax.set_ylabel(f"{direction} is Better KPIs", fontsize=12)

        y_pos = []
        y_labels = []
        x_vals = []
        colors = []

        for i, row in subset.iterrows():
            y_pos.append(i)
            y_labels.append(row["Compact Label"])
            x_vals.append(row["Deviation %"])
            colors.append(get_color(row["Target Reached"]))

        ax.plot(x_vals, y_pos, "o-", color="blue", linewidth=2, zorder=2)
        for x, y, c in zip(x_vals, y_pos, colors):
            ax.scatter(x, y, color=c, s=100, edgecolor="black", zorder=3)

        ax.set_title(f"{region} - {direction} is Better")
```

```
        ax.set_yticks(y_pos)
        ax.set_yticklabels(y_labels)
        ax.invert_yaxis()
        ax.axvline(0, color="green", linestyle="--", alpha=0.5)
        ax.set_xlabel("Deviation %")
        ax.grid(True, axis="x", linestyle="--", alpha=0.5)

        if direction == "High":
            ax.axvspan(-0.10, 0, color="orange", alpha=0.05)
            ax.axvline(-0.10, color="red", linestyle="-.", alpha=0.5)
        else:
            ax.axvspan(0, 0.10, color="orange", alpha=0.05)
            ax.axvline(0.10, color="red", linestyle="-.", alpha=0.5)

        ax.set_xlim(-0.6, 0.6)

fig.suptitle("GEAR Balanced Scorecard - Worm Diagrams by Market\n(Separated by␣
 ↪Optimization Direction)", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])

x_legend_patches = [
    mpatches.Patch(color="green", label="Target Reached"),
    mpatches.Patch(color="orange", label="Close to Target (within ±10%)"),
    mpatches.Patch(color="red", label="Target Not Reached"),
]
fig.legend(handles=x_legend_patches, loc="lower center", bbox_to_anchor=(0.5, 0.
 ↪01), ncol=3)

plt.show()
```



2.5 Q1.5) Discuss the potential advantages and challenges associated with implementing this supplementary graphical visualization for market performance evaluation.

The diagram makes it easy to compare KPI performance across markets at a glance and quickly spot which areas need improvement. It's more intuitive than raw tables and highlights patterns effectively.

However, splitting the chart into "High is Better" and "Low is Better" was necessary for clarity, as combining both would be visually confusing. A fixed ±10% threshold was used for simplicity, while the exam chart applied dynamic thresholds tailored to each KPI. This can be problematic, especially with percentage-based KPIs—it's unclear if 10% means absolute points or a proportional change (e.g. 20% ±10% = 18–22%).

Although the chart could be refined further (e.g. offsetting colored markers to the left), the current version prioritizes simplicity. Finally, since perspectives measure different aspects, the diagram reveals performance patterns, not trends. It's useful for comparing markets side-by-side but not for analyzing changes over time.

3 Exercise 2

[5]:
```
df_ex2 = pd.read_csv("Exercise2.csv")
#df_ex2.info()
#df_ex2.head()
```

3.1 Q2.1) Identify the dependent variable in this context. Which features should be included as independent variables, and why?

By looking at the table provided in the Exam paper, I derive the following:

Dependent Variable:
- `Defect` - This is the target variable we aim to predict (Yes/No).

Independent Variables:
- Include: `Frame`, `Drivetrain`, `Suspension`, `WheelType`, `WheelSize`, `Tires`, `Brake`, `Gear`, `Handlebars`, `Seat`, `Pedals`, `Graphics`, `PaintJob`

- Exclude: `Return`, `Warranty`

The selected independent variables represent the product's configuration that may contribute to defects. `Return` and `Warranty` should be excluded, as they occur **after production** and would introduce future information, leading to data leakage.

3.2 Q2.2) Find the decision tree that maximizes GEAR's expected profits. Thereby, consider a reasonable range of parameters. Choose a cross-validation method where the testing accuracy has a low variance. Explain your answer. Remember to use the parameter option random_state.

[6]:
```
encode = LabelEncoder()

df_ex2["Defect_n"] = encode.fit_transform(df_ex2["Defect"])
df_ex2["Frame_n"] = encode.fit_transform(df_ex2["Frame"])
df_ex2["Drivetrain_n"] = encode.fit_transform(df_ex2["Drivetrain"])
df_ex2["Suspension_n"] = encode.fit_transform(df_ex2["Suspension"])
df_ex2["WheelType_n"] = encode.fit_transform(df_ex2["WheelType"])
df_ex2["Tires_n"] = encode.fit_transform(df_ex2["Tires"])
df_ex2["Brake_n"] = encode.fit_transform(df_ex2["Brake"])
df_ex2["Gear_n"] = encode.fit_transform(df_ex2["Gear"])
df_ex2["Handlebars_n"] = encode.fit_transform(df_ex2["Handlebars"])
df_ex2["Seat_n"] = encode.fit_transform(df_ex2["Seat"])
df_ex2["Pedals_n"] = encode.fit_transform(df_ex2["Pedals"])
df_ex2["Graphics_n"] = encode.fit_transform(df_ex2["Graphics"])
df_ex2["PaintJob_n"] = encode.fit_transform(df_ex2["PaintJob"])
```

[7]:
```
y = df_ex2["Defect_n"]
```

```
X = df_ex2[["Frame_n", "Drivetrain_n", "Suspension_n", "WheelType_n",␣
 ↪"WheelSize", "Tires_n",
            "Brake_n", "Gear_n", "Handlebars_n", "Seat_n", "Pedals_n",␣
 ↪"Graphics_n", "PaintJob_n"]].copy()

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,␣
 ↪random_state=42)
```

[8]:
```
rs=42

param_grid = {
    "max_depth": range(1, 21),
    "ccp_alpha": [0.0002, 0.0003, 0.0004],
}


profit_matrix = [[10000, 9500],[2000, 9100]]

#custom payoff function
def payout(y_true, y_pred):
    matrix = confusion_matrix(y_true, y_pred, labels=[0, 1])
    payoff = np.sum(np.multiply(matrix, profit_matrix))/ len(y_pred)
    return payoff

my_score = make_scorer(payout, greater_is_better=True)


grid = GridSearchCV(
    DecisionTreeClassifier(random_state=rs),
    param_grid,
    verbose=1,
    scoring=my_score,
    cv=10,
)

grid.fit(X_train, y_train)

best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
test_acc = accuracy_score(y_test, y_pred)

cv_results = pd.DataFrame(grid.cv_results_)
mean_profit = grid.best_score_
std_profit = cv_results.loc[cv_results["rank_test_score"] == 1,␣
 ↪"std_test_score"].values[0]

print("Best max_depth:", grid.best_params_["max_depth"])
```

```
print(f"CV Mean Profit per Prediction: {mean_profit:.2f}")
print(f"CV Std (Variance): {std_profit:.2f}")
print(f"Test Accuracy: {test_acc:.4f}")
```

```
Fitting 10 folds for each of 60 candidates, totalling 600 fits
Best max_depth: 8
CV Mean Profit per Prediction: 9908.76
CV Std (Variance): 18.53
Test Accuracy: 0.9903
```

[9]:
```
grid.best_estimator_
```

[9]:
```
DecisionTreeClassifier(ccp_alpha=0.0002, max_depth=8, random_state=42)
```

1. Profit Matrix To evaluate the model based on business impact rather than classification accuracy, I constructed a **profit matrix** using the values provided in the exam:

Actual \ Predicted	Predicted: No Defect	Predicted: Defect
Actual: No Defect (0)	USD 10,000	USD 9,500
Actual: Defect (1)	USD 2,000	USD 9,100

- **False Negatives (missed defects)** are the most costly.
- **True Positives and False Positives** still retain most of the value, with some inspection/repair cost deducted.

---

2. Payoff Function I defined a **custom payoff function** that computes the total profit per prediction using the confusion matrix and the profit matrix. This function was wrapped into a `make_scorer()` object and used as the scoring metric during training.

```
def payout(y_true, y_pred):
    matrix = confusion_matrix(y_true, y_pred, labels=[0, 1])
    payoff = np.sum(np.multiply(matrix, profit_matrix)) / len(y_pred)
    return payoff

my_score = make_scorer(payout, greater_is_better=True)
```

---

3. Grid Search with Cross-Validation (`max_depth` & `ccp_alpha`) To tune the decision tree, I used `GridSearchCV` with a **10-fold cross-validation** and the custom profit-based scoring metric.

- Tuned parameters:
  - `max_depth`: from 1 to 21
  - `ccp_alpha`: [0.0002, 0.0003, 0.0004]
- `random_state=42` was used for reproducibility.
- 10-fold CV was chosen to reduce the variance and provide a stable estimate of model performance.

Although more values of ccp_alpha could have been tested, the best value was found within this range. I limited it to three options to reduce computation time and ensure the code ran faster.

---

4. Model Fit & Results The model was trained on the best combination of parameters found in the grid search.

Best model results: - `max_depth`: 8 - **Mean CV profit per prediction**: USD 9908.76 - **Standard deviation (CV)**: USD 18.53 - **Test accuracy**: 99.03%

This shows the model performs consistently and yields high profit, making it a reliable choice for GEAR"s decision-making process.

3.3 Q2.3) According to your decision tree analysis, which features are most important for explaining whether a product has a defect?

[10]:
```
important_features = pd.DataFrame({"Feature": X_train.columns, "Importance":␣
↪best_model.feature_importances_}).sort_values(by="Importance",␣
↪ascending=False)
important_features
```

[10]:

	Feature	Importance
1	Drivetrain_n	0.550698
3	WheelType_n	0.171034
0	Frame_n	0.137723
4	WheelSize	0.073943
9	Seat_n	0.033845
10	Pedals_n	0.032757
2	Suspension_n	0.000000
5	Tires_n	0.000000
6	Brake_n	0.000000
7	Gear_n	0.000000
8	Handlebars_n	0.000000
11	Graphics_n	0.000000
12	PaintJob_n	0.000000

The most important features for explaining whether a product has a defect are:

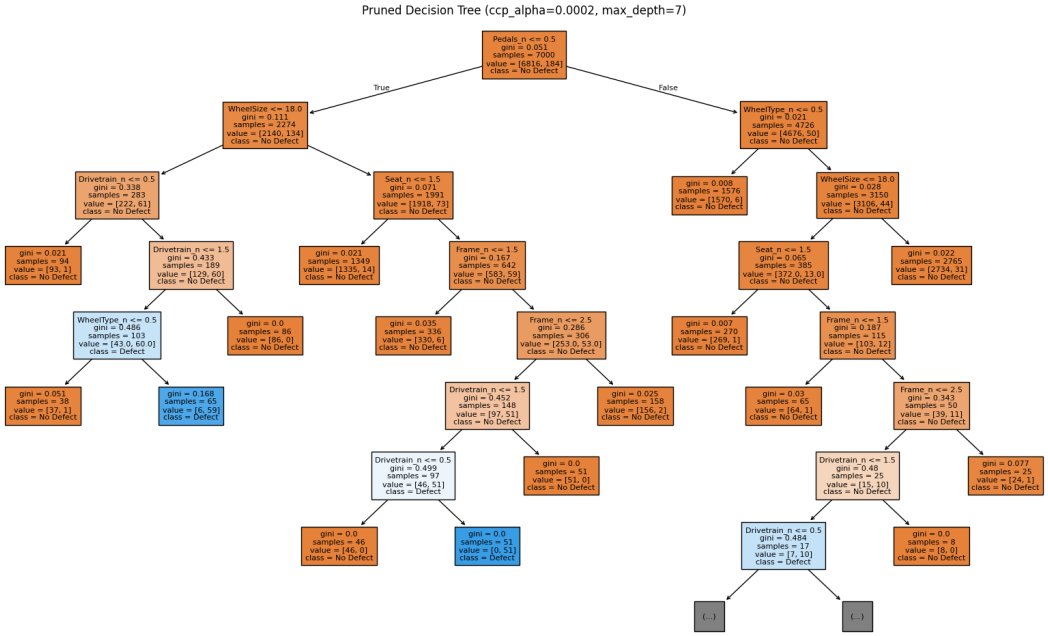
1. **Drivetrain**
2. **WheelType**
3. **FrameType**

These variables received the highest importance scores in the decision tree model, indicating they contribute the most to determining defect status.

3.4 Q2.4) If your decision tree has a maximum depth of less than 8, visualize the decision tree as is. If the maximum depth is 8 or more, visualize the decision tree but prune it to a maximum depth of 7.

Since the unpruned tree reached a `max_depth` of 8, pruning is necessary to prevent overfitting by removing unnecessary or weak branches. We apply the `ccp_alpha` value obtained in Q2.2 to build a pruned decision tree. This helps simplify the model while retaining its predictive power.

[11]:
```
pruned_tree = DecisionTreeClassifier(random_state=rs, ccp_alpha=best_model.
 ↪ccp_alpha)
pruned_tree.fit(X_train, y_train)

plt.figure(figsize=(21, 12))
plot_tree(pruned_tree, feature_names=X.columns, class_names=["No Defect",␣
 ↪"Defect"],
          filled=True, max_depth=7, fontsize=8)
plt.title(f"Pruned Decision Tree (ccp_alpha={best_model.ccp_alpha},␣
 ↪max_depth=7)")
plt.show()
```



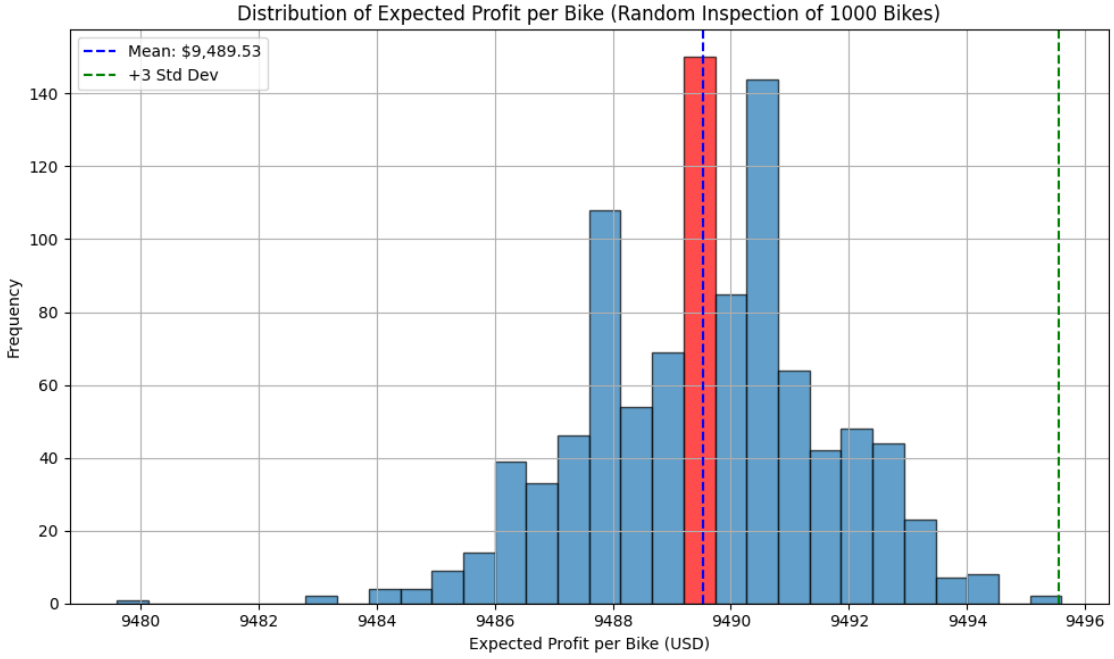
3.5 Q2.5) Provide your recommendation to Jim: Should he conduct a random sample inspection of 1,000 bicycles, or should he apply the insights from your decision tree analysis? Justify your recommendation and explain the economic rationale behind it.

[12]:
```
# Model approach
mean_profit = grid.best_score_
print(f"Mean profit from the model: USD {mean_profit:.2f}")
```

```
Mean profit from the model: USD 9908.76
```

[13]:
```
# Random sample approach
n_bikes = 1000

def random_inspect(n_bikes=n_bikes,
                   defect_rate=y_train.mean(),
                   profit_matrix=[[10000, 9500], [2000, 9100]],
                   n_simulations=1000,
                   random_seed=42):

    np.random.seed(random_seed)
    total_profits = []

    for _ in range(n_simulations):
        defective = np.random.rand(n_bikes) < defect_rate

        n_defective = defective.sum()
        n_non_defective = n_bikes - n_defective

        total_profit = (
            n_defective * profit_matrix[1][1] +
            n_non_defective * profit_matrix[0][1]
        )

        total_profits.append(total_profit)

    return total_profits


simulated_profits_updated = random_inspect(defect_rate=y_train.mean())

profits_array = np.array(simulated_profits_updated)


profits_array_per_bike = profits_array / n_bikes

mean_profit = profits_array_per_bike.mean()
std_profit = profits_array_per_bike.std()
```

```
upper_bound = mean_profit + 3 * std_profit

plt.figure(figsize=(10, 6))
counts, bins, patches = plt.hist(profits_array_per_bike, bins=30,␣
 ↪edgecolor="black", alpha=0.7)

max_count_index = np.argmax(counts)
patches[max_count_index].set_facecolor("red")

plt.axvline(mean_profit, color="blue", linestyle="--", label=f"Mean:␣
 ↪${mean_profit:,.2f}")
plt.axvline(upper_bound, color="green", linestyle="--", label="+3 Std Dev")

plt.title("Distribution of Expected Profit per Bike (Random Inspection of 1000␣
 ↪Bikes)")
plt.xlabel("Expected Profit per Bike (USD)")
plt.ylabel("Frequency")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

print(f"Mean expected profit from randomly inspecting {n_bikes} bikes: USD␣
 ↪{mean_profit:,.2f}")
print(f"Upper bound (+3 std): USD {upper_bound:,.2f}")
```



```
Mean expected profit from randomly inspecting 1000 bikes: USD 9,489.53
Upper bound (+3 std): USD 9,495.55
```

By running the model, we obtain a **model-based profit of USD 9,908.76**, while randomly inspecting 1,000 bikes (using the same defect rate and profit matrix) yields an expected mean profit of **USD 9,495.55**, with a +3 standard deviation upper bound of **USD 9,485.59**. Since the decision tree model clearly outperforms the random inspection strategy, I recommend that Jim select **option (b)** — using insights from the decision tree analysis to guide quality control efforts.

While we could run the simulation additional times to improve reliability, the fact that the model-based profit still exceeds the +3 standard deviation bound of the random strategy, suggests that random inspection is unlikely to be the better option.

---

4 Exercise 3

4.1 Q3.1) How many total login attempts were made in the data set?

[14]:
```
login_data = pd.read_csv("Cybersecurity_case_study_LoginAnalysis_LogFile.csv",␣
 ↪encoding="latin-1")
login_data["Date"] = pd.to_datetime(login_data["Date"])
login_data["LoginTime"] = pd.to_datetime(login_data["LoginTime"], format="%H:%M:
 ↪%S").dt.time
login_data["LoginSuccess"] = login_data["LoginSuccess"] == "Success"


employee_data=pd.read_csv("Cybersecurity_case_study_LoginAnalysis_EmployeeData.
 ↪csv", parse_dates=["StartDate","EndDate"])
```

[15]:
```
total_logins = len(login_data)
print(f"Total login attempts: {total_logins}")
```

```
Total login attempts: 1292180
```

4.2 Q3.2) What percentage of all logins failed?

[16]:
```
successful_logins = login_data["LoginSuccess"].sum()
failed_percentage = (1 - successful_logins / total_logins) * 100
print(f"Percentage of failed logins: {failed_percentage:.2f}%")
```

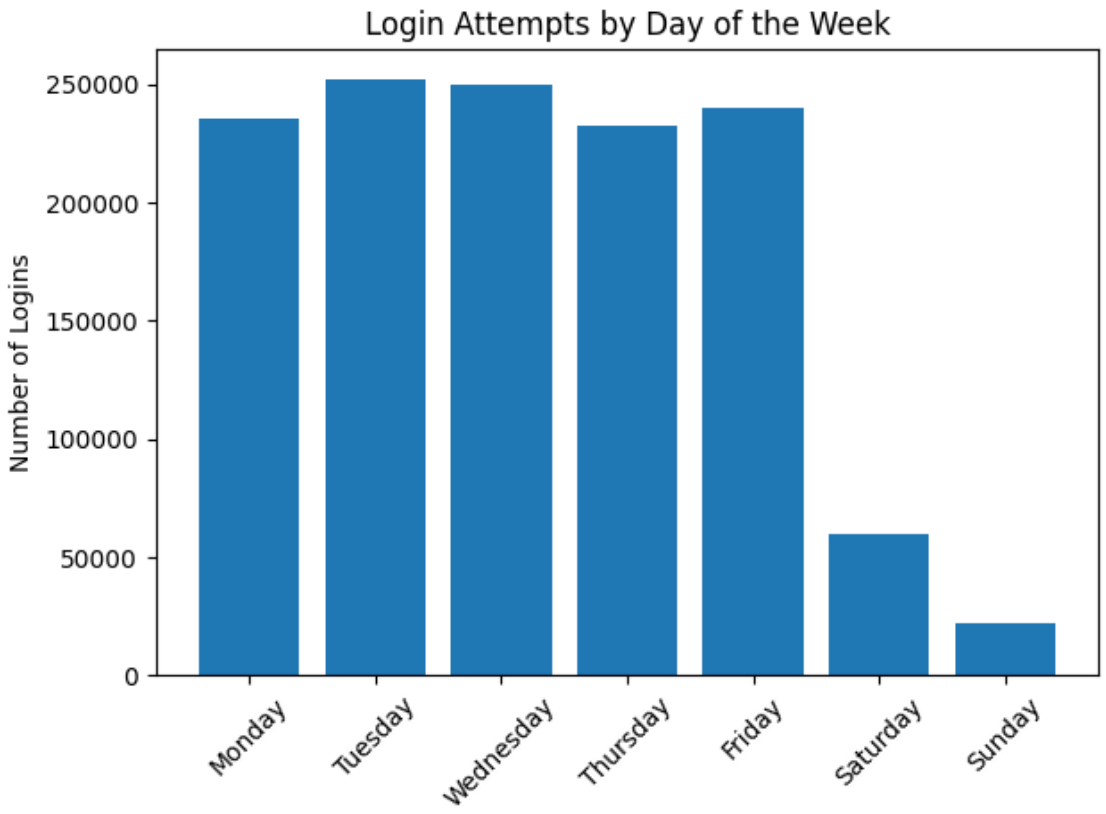
```
Percentage of failed logins: 19.85%
```

4.3 Q3.3) What is the total number of login attempts made on each day of the week (i.e.,Monday, Tuesday)? Does the distribution of logins by day make sense? Why or why not?

[17]:
```
login_data["DayOfWeek"] = login_data["Date"].dt.day_name()
weekday= ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday",␣
 ↪"Sunday"]

login_data["DayOfWeek"] = pd.Categorical(login_data["DayOfWeek"],␣
 ↪categories=weekday, ordered=True)

login_counts = login_data["DayOfWeek"].value_counts().sort_index()

login_counts
```

[17]:
```
DayOfWeek
Monday       235521
Tuesday      252117
Wednesday    249695
Thursday     232287
Friday       240374
Saturday      59952
Sunday        22234
Name: count, dtype: int64
```

[18]:
```
logins_by_day = login_data["DayOfWeek"].value_counts().reindex(weekday)

plt.bar(weekday, logins_by_day.values)
plt.xticks(rotation=45)
plt.ylabel("Number of Logins")
plt.title("Login Attempts by Day of the Week")
plt.tight_layout()
plt.show()
```



Yes, the distribution is consistent with a normal corporate workweek. Higher volumes in the week and drops off during the weekends.

4.4 Q3.4) What is the average number of login attempts an employee makes each day?

[19]:
```
total_logins = len(login_data)

unique_days = login_data["Date"].nunique()

total_users = employee_data["FirstName"].nunique()

avg_logins = total_logins / (unique_days * total_users)

print(f"Average number of login attempts per employee per day: {avg_logins:.
↪2f}")
```

```
Average number of login attempts per employee per day: 4.47
```

4.5 Q3.5) Merge the two datasets. Hint: Make sure you think carefully about how you will join the two data sets before answering the following questions. You want to analyze all employees and all logins, even if there are null values in some fields. Remember null values are not treated as zeros or empty strings.

[20]:
```
#employee_data.head()
```

[21]:
```
#login_data.head()
```

[22]:
```
merged_log_emp=pd.merge(login_data, employee_data, on="UserName", how="outer",␣
↪indicator=True)
merged_log_emp.head()
```

[22]:

	LogEntry	UserName	Date	LoginTime	LoginSuccess	IPAddress	Latitude	Longitude	State	Country	DayOfWeek	FirstName	LastName	StartDate	EndDate	_merge
0	NaN	AAddison89	NaT	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Ashil	Addison	2020-01-02	2020-09-15	right_only
1	5043699.0	AAhlf78	2022-05-02	09:11:12	True	163.116.133.114	41.848297	-87.651703	Illinois	United States	Monday	Adriaens	Ahlf	2018-12-18	NaT	both
2	5043976.0	AAhlf78	2022-05-02	09:34:20	True	163.116.133.114	41.848297	-87.651703	Illinois	United States	Monday	Adriaens	Ahlf	2018-12-18	NaT	both
3	5044480.0	AAhlf78	2022-05-02	10:03:14	True	163.116.133.114	41.848297	-87.651703	Illinois	United States	Monday	Adriaens	Ahlf	2018-12-18	NaT	both
4	5044859.0	AAhlf78	2022-05-02	10:25:34	True	163.116.133.114	41.848297	-87.651703	Illinois	United States	Monday	Adriaens	Ahlf	2018-12-18	NaT	both

I used an outer join to keep all employees and logins, even if some fields are missing. The **indicator=True** flag helps show where each row came from.

4.6 Q3.6) List every employee (one row per employee) who has worked at the company and show the total number of successful logins and the total number of failed logins they have made. Your answer should have four columns labeled in the following order: UserName, EmployeeName (concatenated in one field: FirstName LastName), SuccessfulLogins and FailedLogins. Sort the data so the employees with the most successful logins are listed first

[23]:
```
merged_log_emp["EmployeeName"] = merged_log_emp["FirstName"].fillna("") + " " +␣
 ↪merged_log_emp["LastName"].fillna("")

login_summary = merged_log_emp.groupby(["UserName",␣
 ↪"EmployeeName"])["LoginSuccess"].value_counts().unstack(fill_value=0)

login_summary = login_summary.rename(columns={True: "SuccessfulLogins", False:␣
 ↪"FailedLogins"}).reset_index()

if "SuccessfulLogins" not in login_summary:
    login_summary["SuccessfulLogins"] = 0
if "FailedLogins" not in login_summary:
    login_summary["FailedLogins"] = 0

login_summary = login_summary[["UserName", "EmployeeName", "SuccessfulLogins",␣
 ↪"FailedLogins"]]
login_summary = login_summary.sort_values(by="SuccessfulLogins",␣
 ↪ascending=False)

login_summary.head(10)
```

[23]:

LoginSuccess	UserName	EmployeeName	SuccessfulLogins	FailedLogins
99	CGonnin6	Cullie Gonnin	10491	1074
348	NLax62	Nadeen Lax	9301	985
207	GWhitelock12	Gil Whitelock	8996	2402
402	SBenito53	Son Benito	8481	2235
188	GEmons23	Garrek Emons	8469	3173
53	BFateley41	Baxie Fateley	8291	529
66	BMacLleese55	Babbette MacLleese	8159	2716
206	GWenban49	Georgetta Wenban	7829	1351
223	IAbberley18	Ilyssa Abberley	7796	1163
325	MPhillpot35	Meagan Phillpot	7528	781

4.7 Q3.7) Compute the failure login rate for each employee (computed as failed logins divided by the total login attempts) and sort the data to show employees who most frequently fail at logging in at the top of the list. Only show employees who have attempted at least one login in the data. Your final answer should have these columns in this order: UserName, EmployeeName (concatenated in one field: FirstName LastName) and FailureRate. What problems do you see from this analysis?

[24]:
```
login_summary_rate = login_summary.copy()
login_summary_rate["TotalLogins"] = login_summary_rate["SuccessfulLogins"] +␣
 ↪login_summary_rate["FailedLogins"]
login_summary_rate = login_summary_rate[login_summary_rate["TotalLogins"] > 0]
login_summary_rate["FailureRate"] = login_summary_rate["FailedLogins"] /␣
 ↪login_summary_rate["TotalLogins"]

login_fail_rate = login_summary_rate[["UserName", "EmployeeName",␣
 ↪"FailureRate"]].sort_values("FailureRate", ascending=False)
login_fail_rate.reset_index(drop=True, inplace=True)

login_fail_rate.head(10)
```

[24]:

LoginSuccess	UserName	EmployeeName	FailureRate
0	Admin		1.000000
1	EJeanneau51	Evyn Jeanneau	0.819875
2	ABragger43	Adele Bragger	0.570687
3	LEvennett22	Leyla Evennett	0.533605
4	MVedeniktov67	Margery Vedeniktov	0.474114
5	GDunstone64	Galvin Dunstone	0.470930
6	MCowdray63	Moses Cowdray	0.460048
7	EMaclaine19	Ebba Maclaine	0.451007
8	SSpeariett40	Stafford Speariett	0.439394
9	EDoveston85	Emelina Doveston	0.414218

The table shows that the **Admin** account has a 100% failure rate, which likely means it is no longer in use or has been misconfigured. Some users are attempting to log in but are consistently failing. In particular, three users have failure rates above 50%, which could point to their accounts being compromised or targeted by brute-force attacks. This should be investigated further. The system should also record why login attempts fail and whether the same user is logging in from different IP addresses. While the table doesn't explain the reasons, it clearly highlights users who may need to be reviewed more closely.

4.8 Q3.8) Identify employees who attempted to log in a day or longer (i.e., not the same day) after they were terminated from working at the organization. The final table should have the following columns in this order: LogEntry, EmployeeName, EndDate, LoginDate and LoginSuccess. Sort the data in ascending order by LogEntry. Based on this analysis, what recommendations would you make to GEAR to strengthen its internal controls?

[25]:
```
merged_log_emp2 = merged_log_emp.copy()

merged_log_emp2 = merged_log_emp2.dropna(subset=["EndDate"])

merged_log_emp2["Date"] = pd.to_datetime(merged_log_emp2["Date"])
merged_log_emp2["EndDate"] = pd.to_datetime(merged_log_emp2["EndDate"])

logged_after = merged_log_emp2[merged_log_emp2["EndDate"] <␣
 ↪merged_log_emp2["Date"]]


user_terminated = logged_after[["LogEntry", "EmployeeName", "EndDate", "Date",␣
 ↪"LoginSuccess"]].sort_values("LogEntry", ascending=True)
user_terminated = user_terminated.rename(columns={"Date": "LoginDate"})
user_terminated
```

[25]:

	LogEntry	EmployeeName	EndDate	LoginDate	LoginSuccess
777137	5046603.0	Kameko Mathwen	2013-09-14	2022-05-02	True
777138	5046729.0	Kameko Mathwen	2013-09-14	2022-05-02	False
777139	5046733.0	Kameko Mathwen	2013-09-14	2022-05-02	False
777140	5046741.0	Kameko Mathwen	2013-09-14	2022-05-02	True
777141	5047269.0	Kameko Mathwen	2013-09-14	2022-05-02	False
…	…	…	…	…	…
641691	6334541.0	Hugues MacMenamin	2022-11-18	2022-11-30	False
641692	6334559.0	Hugues MacMenamin	2022-11-18	2022-11-30	True
641693	6334574.0	Hugues MacMenamin	2022-11-18	2022-11-30	True
641694	6334576.0	Hugues MacMenamin	2022-11-18	2022-11-30	True
641695	6334616.0	Hugues MacMenamin	2022-11-18	2022-11-30	True

[4595 rows x 5 columns]

The data shows multiple login attempts by employees after their termination dates, some of which were successful. This suggests that former employees credentials may remain active even after they leave the organization. It appears that the system does not automatically lock out users when an EndDate is present. One way to improve this would be to introduce a separate variable that marks users as active or inactive based on their employment status. This flag could then be used to prevent access for terminated employees. Additionally, implementing alerts for login attempts made by terminated users would further enhance security and help detect unauthorized access attempts in real time.

4.9 Q3.9) How many attempted logins were made that do not correspond to an employee listed in the employee file? How many of these attempts were successful vs. unsuccessful? From which countries were these login attempts made? Provide any insight into what you believe is happening with these logins.

[26]:
```
no_name_logins=merged_log_emp[merged_log_emp["FirstName"].isnull()]
no_name_logins["UserName"].value_counts()
```

[26]:
```
UserName
Admin    1277
Name: count, dtype: int64
```

[27]:
```
no_name_logins.groupby("UserName")["Country"].value_counts().head(10)
```

[27]:
```
UserName  Country
Admin     China                220
          Canada               168
          Germany              159
          United Kingdom        96
          Taiwan                63
          Australia             56
          Hong Kong S.A.R.      55
          South Korea           51
          Netherlands           42
          Italy                 40
Name: count, dtype: int64
```

[28]:
```
no_name_logins.groupby("UserName")["LoginSuccess"].value_counts().head(10)
```

[28]:
```
UserName  LoginSuccess
Admin     False           1277
Name: count, dtype: int64
```

There were a total of 1,277 login attempts that did not correspond to any employee listed in the employee file. None of these logins were successful, resulting in a 100% failure rate. The attempts primarily originated from China, Canada, and Germany.

These patterns suggest that malicious actors may be attempting to access the system using common usernames such as admin. The admin account, in particular, shows a high number of failed login attempts. Given its 100% failure rate and absence from the employee records, it appears that the admin account is either not in use or does not exist in the system. This behavior indicates possible brute-force or credential-stuffing attempts, and further monitoring or IP-based blocking may be warranted.