[STAT-315] Bikeshare Insights Data Analysis Project

Defining the questions:

We first ask the following questions:

1. Can we predict which casual riders are most likely to benefit from a membership, enabling targeted promotional strategies?
2. Are there any seasonal or temporal patterns in ridership behavior that could be used to optimize station positioning and bike allocations to stations?
3. Which factors (such as membership status, trip length, bike type, day of week, or station location) most strongly influence whether a rider chooses an electric versus a classic bike, and how much do these factors impact overall demand?

[STAT-315] Bikeshare Insights Data Analysis Project

GitHub repository: https://github.com/leon934/stat-315-final-project

Defining the questions:

We first ask the following questions:

1. Can we predict which casual riders are most likely to benefit from a membership, enabling targeted promotional strategies?
2. Are there any seasonal or temporal patterns in ridership behavior that could be used to optimize station positioning and bike allocations to stations?
3. Which factors (such as membership status, trip length, bike type, day of week, or station location) most strongly influence whether a rider chooses an electric versus a classic bike, and how much do these factors impact overall demand?

Data collection

For our given questions, we decide to leverage the Divvy dataset previously used for our mini-project. Simply run the following cell to obtain the bike sharing insights data for the year of 2023. It will be stored in `./data/`.

In [ ]:
```
!python combine.py
```

```
Downloading and unzipping all Divvy .csv files.: 100%|█| 12/12 [00:25<00:00,
2.
Reading all Divvy .csv files.: 100%|████████████| 12/12 [00:16<00:00,  1.36
s/it]
Creating concatenated .csv file.
Successfully created merged .csv file. Path is ./data/2023-divvy-tripdata.cs
v
```

Data cleaning and preparation

We then prepare the data for our analysis by cleaning out unusual rows and adding additional features.

In [20]:
```python
# required imports
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [21]:
```python
# loading data
divvy_df = pd.read_csv("./data/2023-divvy-tripdata.csv", index_col=0)
divvy_df = divvy_df.reset_index(drop=True)
```

In [32]:
```python
# get rid of abnormally long or short ride times (<1 minute or >2 hours)
divvy_df["started_at"] = pd.to_datetime(divvy_df["started_at"])
divvy_df["ended_at"] = pd.to_datetime(divvy_df["ended_at"])

divvy_df["ride_duration_min"] = (divvy_df["ended_at"] - divvy_df["started_at

divvy_df = divvy_df[(1 <= divvy_df["ride_duration_min"]) & (divvy_df["ride_c
```

In [33]:
```python
# get rid of abnormally long distances
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth radius in kilometers
    lat1_rad, lon1_rad = np.radians(lat1), np.radians(lon1)
    lat2_rad, lon2_rad = np.radians(lat2), np.radians(lon2)
    dlat = lat2_rad - lat1_rad
    dlon = lon2_rad - lon1_rad

    a = np.sin(dlat / 2.0) ** 2 + np.cos(lat1_rad) * np.cos(lat2_rad) * np.s
    c = 2 * np.arcsin(np.sqrt(a))
    return R * c

divvy_df["distance_km"] = haversine(
    divvy_df["start_lat"],
    divvy_df["start_lng"],
    divvy_df["end_lat"],
    divvy_df["end_lng"]
)

divvy_df = divvy_df[(divvy_df["distance_km"] <= 15) & (divvy_df["distance_km
```

In [34]:
```python
# adds season to dataframe
def season(month: int):
    if month in [1, 2, 12]:
```

```
        return "Winter"
    elif month in [3, 4, 5]:
        return "Spring"
    elif month in [6, 7, 8]:
        return "Summer"
    else:
        return "Fall"

divvy_df["hour"] = divvy_df["started_at"].dt.hour
divvy_df["dayofweek"] = divvy_df["started_at"].dt.day_name()
divvy_df["month"] = divvy_df["started_at"].dt.month
divvy_df["season"] = divvy_df["started_at"].dt.month.apply(season)
```

In [35]:
```
# add km/hr to dataset, removing abnormally low values
# they have significantly higher ride times (~1 hr)
divvy_df["avg_velocity_km_per_hr"] = divvy_df["distance_km"] / (divvy_df["ri
divvy_df = divvy_df[divvy_df["avg_velocity_km_per_hr"] > 2]
```

In [26]:
```
# ensure no null values in dataframe
divvy_df = divvy_df.dropna()
```

In [27]:
```
# get both start and end stations with non-matching names and ids
# e.g. one station is mapped to two ids on either start or end
stations = pd.concat([divvy_df[["start_station_name", "start_station_id"]].r
                      divvy_df[["end_station_name", "end_station_id"]].renam
                      ignore_index=True)

name_to_id = (
    stations.groupby("station_name")["station_id"]
      .agg(lambda x: x.value_counts().idxmax())
)

mismatched_start_stations = divvy_df["start_station_id"] != divvy_df["start_
mismatched_end_stations = divvy_df["end_station_id"] != divvy_df["end_statio

suspicious_stations = pd.concat([divvy_df[mismatched_start_stations]["start_
divvy_df = divvy_df[~divvy_df["start_station_name"].isin(suspicious_stations
```

Data analysis

Question 1. Can we predict which casual riders are most likely to benefit from a membership, enabling targeted promotional strategies?

We first create additional features such as factoring in time by separating the timestamps into month and days, then cyclically encoding their values to better capture the relationship. Also, some columns are dropped such as the station name and ID, since there is no meaningful information from using those as they're captured in the longitude and latitude.

In [36]:
```
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import Pipeline

from sklearn.compose import ColumnTransformer
```

```python
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTra

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn import svm
from sklearn.calibration import CalibratedClassifierCV

from xgboost import XGBClassifier

from catboost import CatBoostClassifier
```

In [37]:

```python
ml_divvy_df = divvy_df.copy()
ml_divvy_df = ml_divvy_df.drop(columns=["dayofweek"])

# obtains only cols that can be used for features in the model
# creates the target column
Y_df = divvy_df["member_casual"]
Y_df = Y_df.map({
    "member": 1,
    "casual": 0
})

# columns to drop after creating new features
drop_cols = [
    "ride_id",
    "started_at", "ended_at",
    "start_station_name", "start_station_id",
    "end_station_name", "end_station_id",
    "avg_velocity_km_per_hr",
    "member_casual"
]

def encode_cyclical(col, period):
    return FunctionTransformer(lambda x: np.hstack([
        np.sin(2 * np.pi * x / period),
        np.cos(2 * np.pi * x / period)
    ]))

# generates new features from date
ml_divvy_df["month"] = divvy_df["started_at"].dt.month
ml_divvy_df["day"] = divvy_df["started_at"].dt.day
ml_divvy_df["weekday"] = divvy_df["started_at"].dt.weekday

# columns to process in transformer
one_hot_encode_cols = ["rideable_type", "season", "weekday"]
numeric_cols = ["start_lat", "start_lng", "end_lat", "end_lng", "ride_durati
```

We then apply transformations such as one-hot-encoding certain categorical variables as well as drop the meaningless columns.

Once the final dataset is obtained, we then fit them to various models and obtain the ROC-AUC curve, since we care about ranking casual users based on how "member-like" they are. We use cross-validation to utilize the same dataset to find the best model in terms of our desired metric.

In [38]:
```
# creates one-hot-encoded representations and removes improper data
preprocessor = ColumnTransformer(
    transformers=[
        ("one_hot_encode", OneHotEncoder(), one_hot_encode_cols),
        ("month_cyclical", encode_cyclical("month", 12), ["month"]),
        ("day_cyclical", encode_cyclical("day", 31), ["day"]),
        ("weekday_cyclical", encode_cyclical("weekday", 7), ["weekday"]),
        ("scale", StandardScaler(), numeric_cols),
        ("drop", "drop", drop_cols),
    ],
    remainder="passthrough"
)

SEED = 1

models = {
    "LogReg": LogisticRegression(
        max_iter=5000,
        solver="liblinear",
        penalty="l2",
        random_state=SEED
    ),
    "Support Vector Machine (SVM)": svm.LinearSVC(),
    "XGBoost": XGBClassifier(
        n_estimators=300,
        learning_rate=0.1,
        max_depth=6,
        subsample=0.9,
        colsample_bytree=0.9,
        eval_metric="logloss",
        tree_method="hist",
        random_state=SEED
    ),
    "CatBoost": CatBoostClassifier(
        iterations=300,
        depth=6,
        learning_rate=0.1,
        l2_leaf_reg=3.0,
        loss_function="Logloss",
        bootstrap_type="Bayesian",
        verbose=False,
        random_state=SEED
    ),
    "Random Forest": RandomForestClassifier(
        n_estimators=150,
        max_depth=20,
```

```
        min_samples_split=20,
        min_samples_leaf=10,
        max_features="sqrt",
        max_samples=0.7,
        class_weight="balanced",
        random_state=SEED
    ),
}
```

Once we've obtained the models we want to evaluate, we then iterate through each to determine which results in the best ROC-AUC value using cross validation with 5 folds.

In [ ]:
```
model_scores = []

for i, (name, model) in enumerate(models.items()):
    pipeline = Pipeline([
        ("prep", preprocessor),
        ("classifier", model)
    ])

    score = cross_val_score(
        pipeline,
        ml_divvy_df, Y_df,
        cv=5,
        scoring="roc_auc"
    )

    print(f"{name}:")
    print(f"Average ROC-AUC score:\n{score.mean():.4f}")
    print()
```

Then once each training pipeline finishes, we obtain the average ROC-AUC score among each model, where we see that SVMs perform the best. Thus, we will use this model for further exploratory analysis.

Model	Average ROC-AUC
Support Vector Machine (SVM)	**0.6232**
XGBoost	0.5992
Logistic Regression (LogReg)	0.6210
CatBoost	0.6152
Random Forest	0.5863

In [ ]:
```
X_casual_df = ml_divvy_df[ml_divvy_df["member_casual"] == "casual"].drop(col

# retrain model
svm_model = svm.LinearSVC()
model = CalibratedClassifierCV(svm_model, cv=5)

best_pipeline = Pipeline([
    ("prep", preprocessor),
```

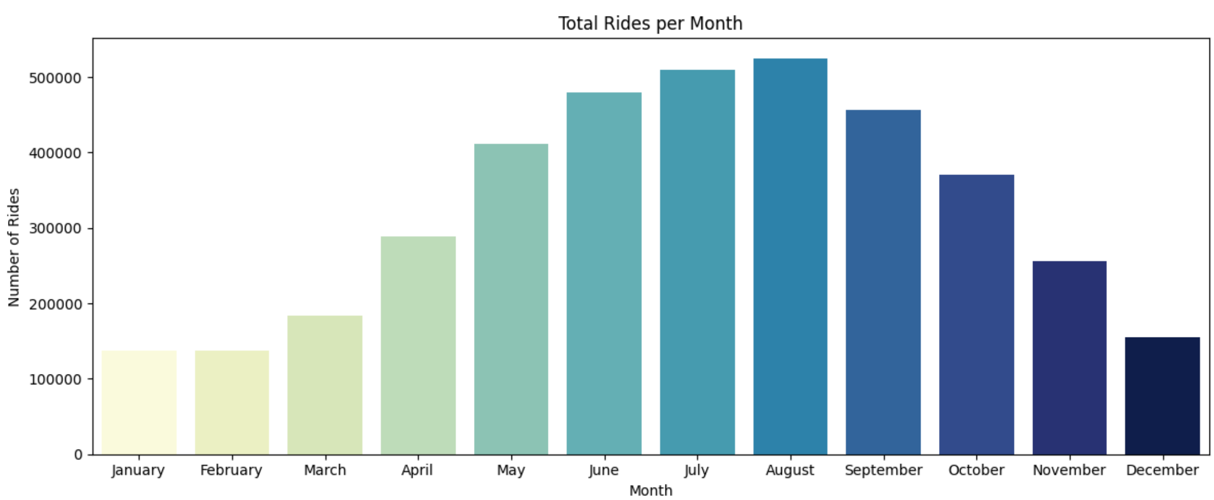
```
    ("classifier", model)
])

best_pipeline.fit(ml_divvy_df, Y_df)

probabilities = pd.DataFrame(best_pipeline.predict_proba(X_casual_df)[:, 1])

candidates_df = X_casual_df.copy()
candidates_df["member_probability"] = probabilities

top_5_prob = probabilities.quantile(0.95).item()

top_candidates_df = candidates_df[candidates_df["member_probability"] >= top
bot_candidates_df = candidates_df[candidates_df["member_probability"] < top_
```

Q2: Are there any seasonal or temporal patterns in ridership behavior that could be used to optimize station positioning and bike allocations to stations?

Analysis #1: Monthly and Seasonal Ridership Volume

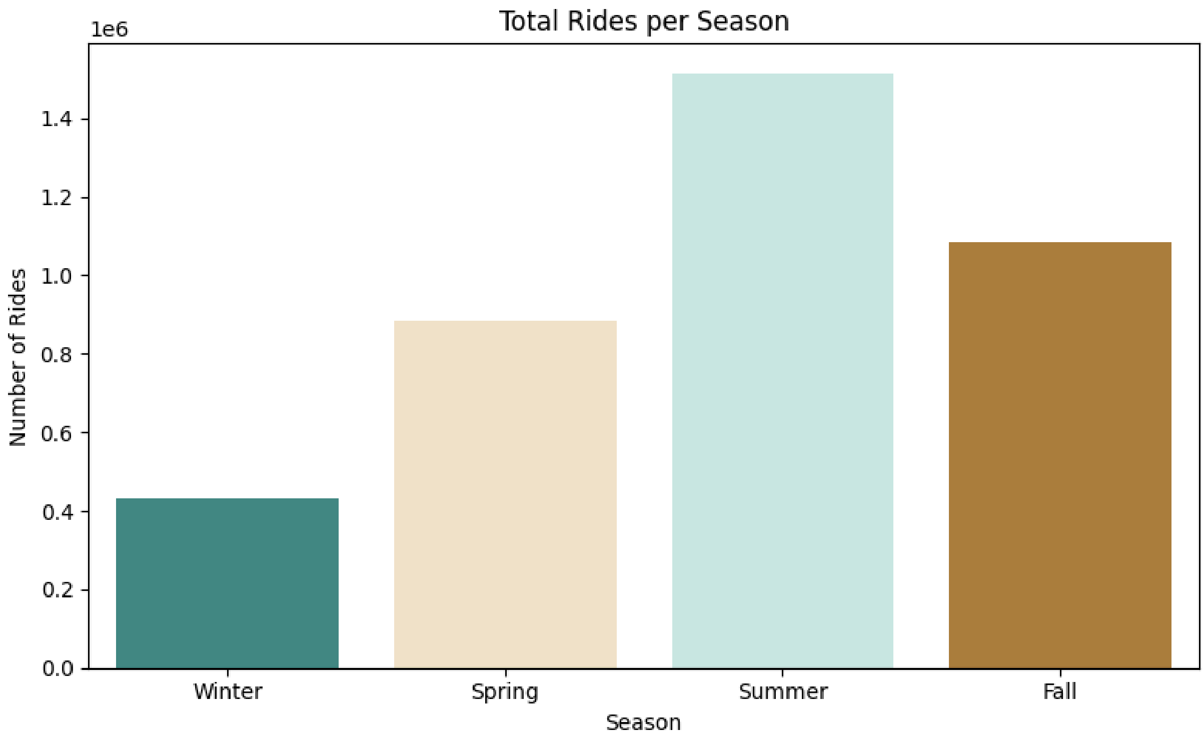
In [29]:
```
import seaborn as sns
```

In [243…
```
# total number of rides per month
month_map = ["January", "February", "March", "April", "May", "June", "July",
monthly_counts = divvy_df.groupby("month").size().reset_index(name="ride_cou

plt.figure(figsize=(12, 5))
sns.barplot(data=monthly_counts, x=month_map, y="ride_count", hue="month", p
plt.title("Total Rides per Month")
plt.xlabel("Month")
plt.ylabel("Number of Rides")
plt.tight_layout()
plt.show()
```



In [40]:
```
# total number of riders by season
season_order = ["Winter", "Spring", "Summer", "Fall"]
seasonal_counts = (
    divvy_df
    .groupby("season")
```

```
    .size()
    .reset_index(name="ride_count")
)

plt.figure(figsize=(8, 5))
sns.barplot(data=seasonal_counts, x="season", y="ride_count", order=season_c
plt.title("Total Rides per Season")
plt.xlabel("Season")
plt.ylabel("Number of Rides")
plt.tight_layout()
plt.show()
```

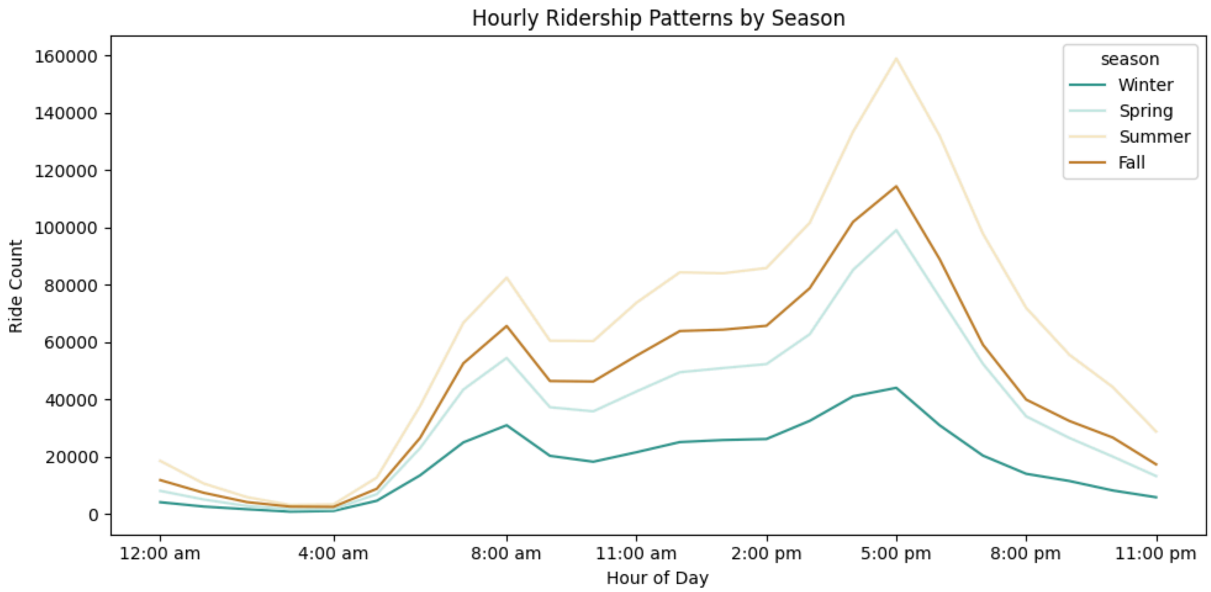


Analysis #2: Hourly and Daily Ridership Volume

In [41]:

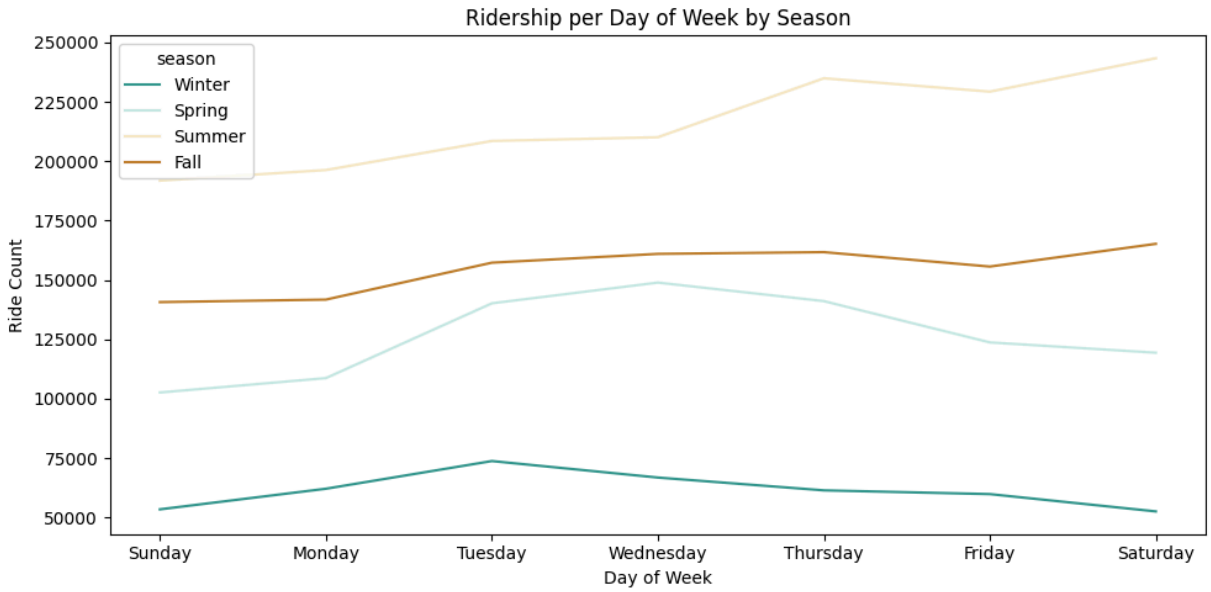
```
hour_season_counts = (
    divvy_df
    .groupby(["season", "hour"])
    .size()
    .reset_index(name="ride_count")
)

plt.figure(figsize=(10, 5))
ax = sns.lineplot(
    data=hour_season_counts,
    x="hour",
    y="ride_count",
    hue="season",
    hue_order=season_order,
    palette="BrBG_r"
)
ax.set_xticks([0, 4, 8, 11, 14, 17, 20, 23])
ax.set_xticklabels(["12:00 am","4:00 am","8:00 am","11:00 am","2:00 pm","5:0
```

```
plt.title("Hourly Ridership Patterns by Season")
plt.xlabel("Hour of Day")
plt.ylabel("Ride Count")
plt.tight_layout()
plt.show()
```



In [42]:
```
day_order = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday

# enforce ordering
divvy_df["dayofweek"] = pd.Categorical(divvy_df["dayofweek"], categories=day

daily_counts = (
    divvy_df
    .groupby(["season", "dayofweek"])
    .size()
    .reset_index(name="ride_count")
)

plt.figure(figsize=(10, 5))
sns.lineplot(
    data=daily_counts,
    x="dayofweek",
    y="ride_count",
    hue="season",
    hue_order=season_order,
    palette="BrBG_r"
)

plt.title("Ridership per Day of Week by Season")
plt.xlabel("Day of Week")
plt.ylabel("Ride Count")
plt.tight_layout()
plt.show()
```

```
/var/folders/fv/0n5v977559s5d507wcjlym4m0000gn/T/ipykernel_13437/3461163747.
py:7: FutureWarning: The default of observed=False is deprecated and will be
changed to True in a future version of pandas. Pass observed=False to retain
current behavior or observed=True to adopt the future default and silence th
is warning.
  divvy_df
```

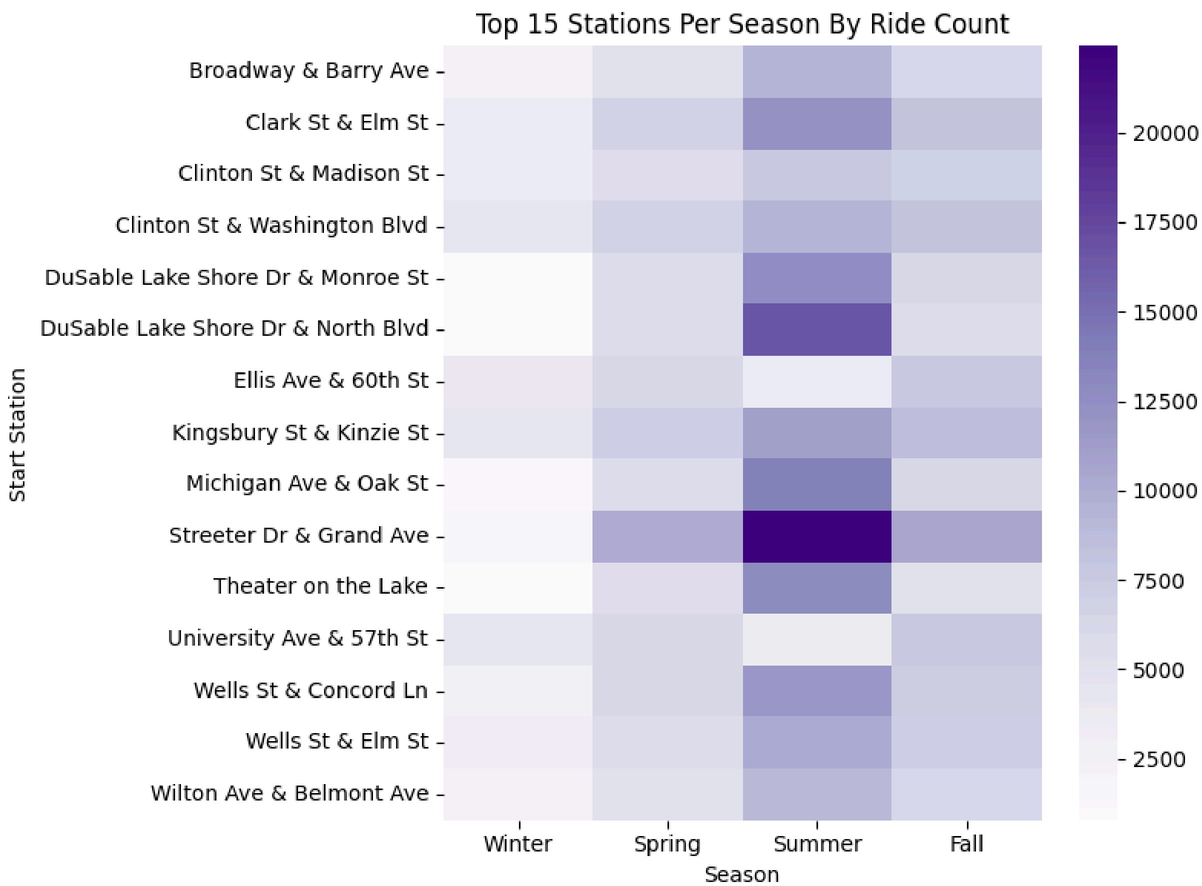


Analysis #3: Station Statistics by Season

In [240…

```
station_season_counts = (
    divvy_df
    .groupby(["season", "start_station_name"])
    .size()
    .reset_index(name="ride_count")
)

overall_top = (
    station_season_counts
    .groupby("start_station_name")["ride_count"]
    .sum()
    .sort_values(ascending=False)
    .head(15)
    .index
)

top_matrix = (
    station_season_counts[
        station_season_counts["start_station_name"].isin(overall_top)
    ]
    .pivot(index="start_station_name", columns="season", values="ride_count"
    .reindex(columns=season_order)
)

plt.figure(figsize=(8, 6))
sns.heatmap(
    top_matrix,
    cmap="Purples"
```

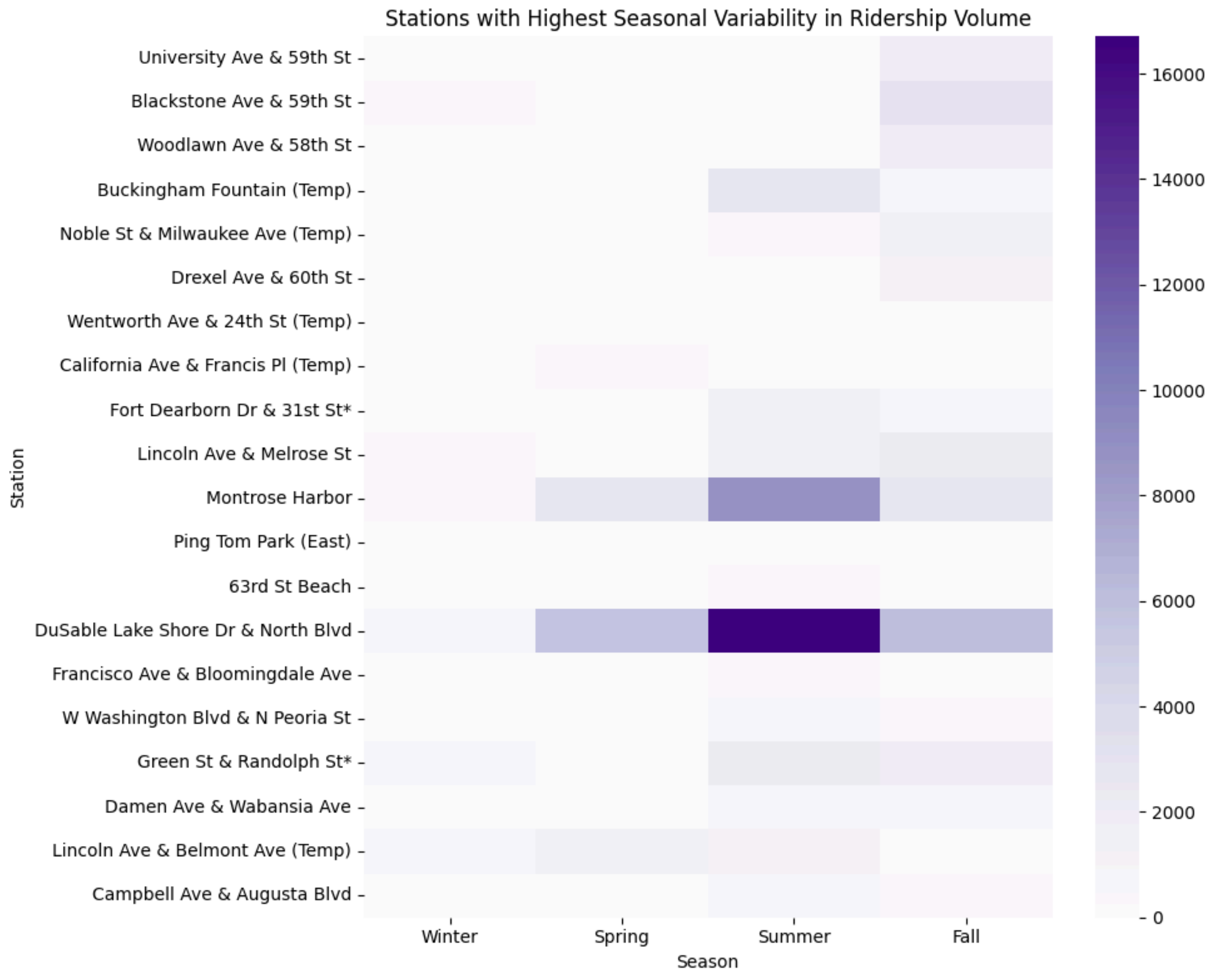
```
)
plt.title("Top 15 Stations Per Season By Ride Count")
plt.xlabel("Season")
plt.ylabel("Start Station")
plt.tight_layout()
plt.show()
```



In [235…

```
station_matrix = station_season_counts.pivot(
    index="start_station_name",
    columns="season",
    values="ride_count"
).fillna(0)

station_matrix["mean"] = station_matrix.mean(axis=1)
station_matrix["std"] = station_matrix.std(axis=1)
station_matrix["cv"] = station_matrix["std"] / station_matrix["mean"]
station_matrix_filtered = station_matrix[station_matrix["mean"] > 50]

top_var = station_matrix_filtered.sort_values("cv", ascending=False).head(20

plt.figure(figsize=(10, 8))
sns.heatmap(
    top_var[season_order],
    cmap="Purples"
)
plt.title("Stations with Highest Seasonal Variability in Ridership Volume")
plt.xlabel("Season")
plt.ylabel("Station")
```

```
plt.tight_layout()
plt.show()
```

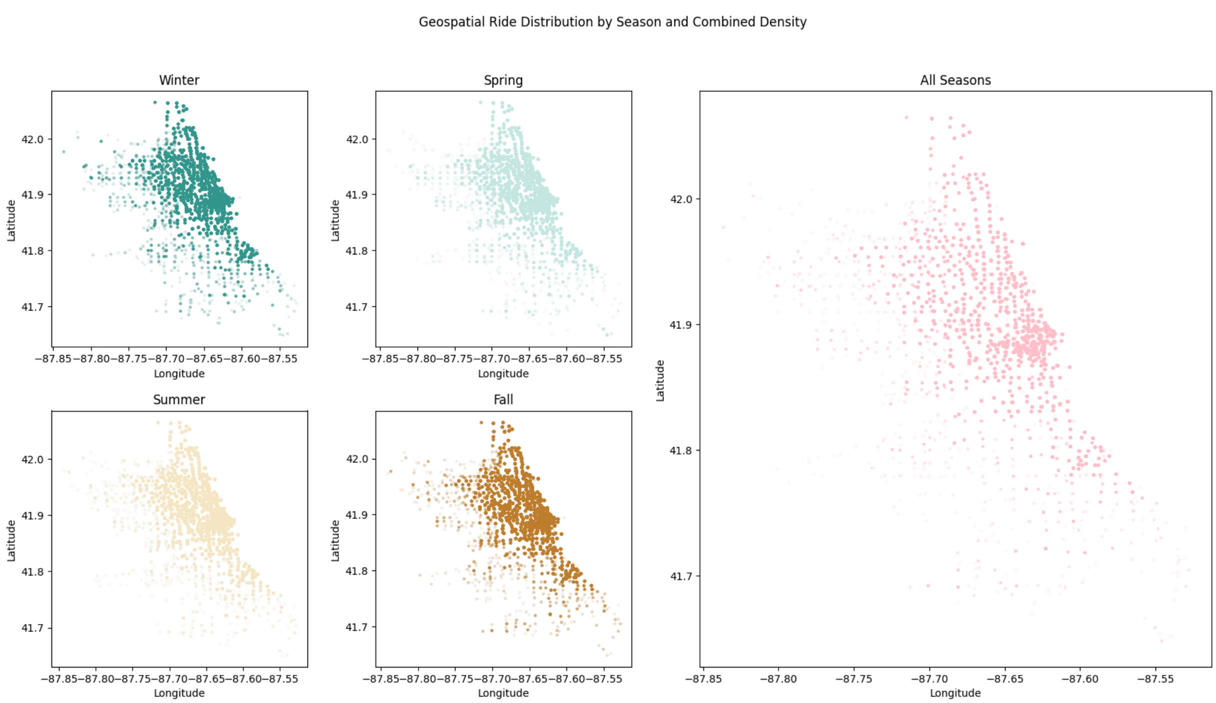


Analysis #4: Ridership Geospatial Distribution by Season

In [242…

```
import matplotlib.gridspec as gridspec

palette = sns.color_palette("BrBG_r", n_colors=4)
season_to_color = dict(zip(season_order, palette))

fig = plt.figure(figsize=(20, 10))
gs = gridspec.GridSpec(2, 3, width_ratios=[1,1,2], wspace=0.2, hspace=0.25)

ax_winter = fig.add_subplot(gs[0, 0])
ax_spring = fig.add_subplot(gs[0, 1])
ax_summer = fig.add_subplot(gs[1, 0])
ax_fall   = fig.add_subplot(gs[1, 1])

axes_season = {
    "Winter": ax_winter,
    "Spring": ax_spring,
    "Summer": ax_summer,
    "Fall": ax_fall
}

for season in season_order:
```

```
    ax = axes_season[season]
    df_s = divvy_df[divvy_df["season"] == season]

    xs = pd.concat([df_s["start_lng"], df_s["end_lng"]])
    ys = pd.concat([df_s["start_lat"], df_s["end_lat"]])

    idx = np.random.choice(xs.index, min(120000, len(xs)), replace=False)

    ax.scatter(xs.loc[idx], ys.loc[idx],
               s=4, alpha=0.08, color=season_to_color[season])
    ax.set_title(season)
    ax.set_xlabel("Longitude")
    ax.set_ylabel("Latitude")

ax_total = fig.add_subplot(gs[:, 2])

xs_all = pd.concat([divvy_df["start_lng"], divvy_df["end_lng"]])
ys_all = pd.concat([divvy_df["start_lat"], divvy_df["end_lat"]])

idx_all = np.random.choice(xs_all.index, min(120000, len(xs_all)), replace=F

ax_total.scatter(xs_all.loc[idx_all], ys_all.loc[idx_all],
                 s=4, alpha=0.06, color="pink")
ax_total.set_title("All Seasons")
ax_total.set_xlabel("Longitude")
ax_total.set_ylabel("Latitude")

plt.suptitle("Geospatial Ride Distribution by Season and Combined Density")
plt.show()
```



Q3: Which factors (such as membership status, trip length, bike type, day of week, or station location) most strongly influence whether a rider chooses an electric versus a classic bike, and how much do these factors impact overall demand?

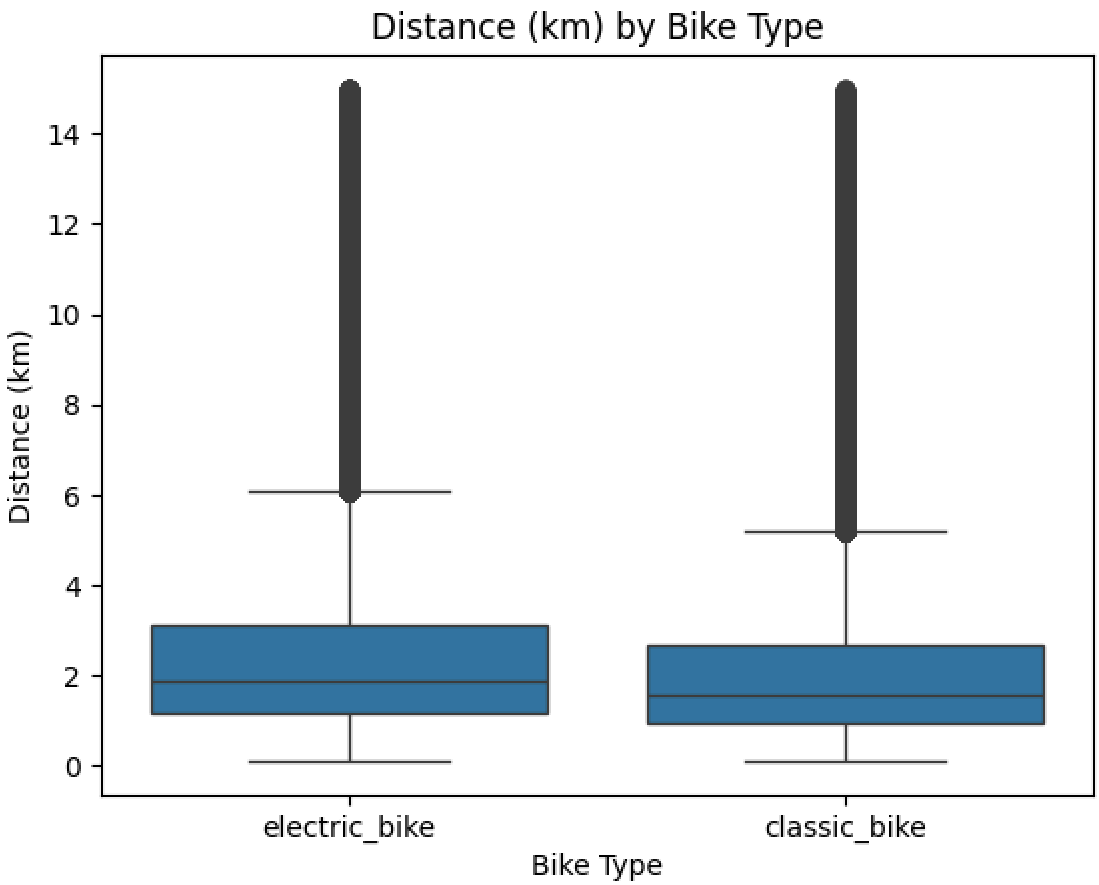
In this section, we investigate which ride and rider characteristics most strongly predict whether a user selects an electric bike or a classic bike. Understanding these patterns can help optimize bike availability, forecast demand for electric bikes, and reveal behavioral differences across membership types, time of day, distance, and season.

In [31]:
```
# Filter to only classic + electric bikes
df_q3 = divvy_df[divvy_df['rideable_type'].isin(['classic_bike', 'electric_b

# Encode target variable (1 = electric)
df_q3['electric'] = (df_q3['rideable_type'] == 'electric_bike').astype(int)

# Convert ride_length (in seconds) to minutes
df_q3['duration_min'] = divvy_df["ride_duration_min"]

# Haversine distance calculation
def haversine(lat1, lon1, lat2, lon2):
    R = 6371  # Earth radius in km
    from math import radians, sin, cos, sqrt, atan2
    dlat = radians(lat2 - lat1)
    dlon = radians(lon2 - lon1)
    a = sin(dlat/2)**2 + cos(radians(lat1)) * cos(radians(lat2)) * sin(dlon/
    c = 2 * atan2(sqrt(a), sqrt(1-a))
    return R * c

# Compute trip distance
df_q3['distance_km'] = df_q3.apply(
    lambda r: haversine(r['start_lat'], r['start_lng'], r['end_lat'], r['enc
    axis=1
)

# Compute trip velocity
df_q3['velocity_kmh'] = df_q3['distance_km'] / (df_q3['duration_min'] / 60)

# Extract time-based features
df_q3['hour'] = pd.to_datetime(df_q3['started_at']).dt.hour
df_q3['day_of_week'] = pd.to_datetime(df_q3['started_at']).dt.dayofweek

# Peak hour flag (typical commute windows)
df_q3['is_peak'] = df_q3['hour'].isin([7,8,9,16,17,18]).astype(int)
```

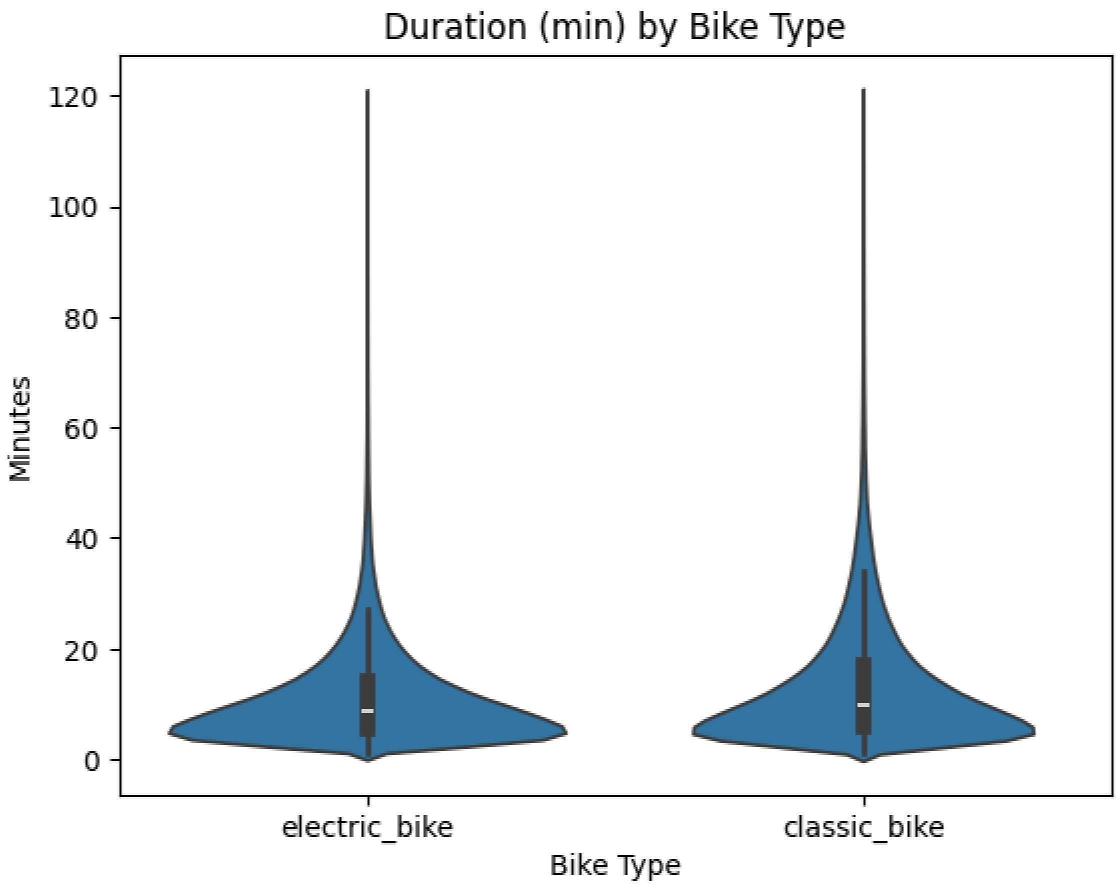
Exploratory Data Visualization

Before modeling, we examine how electric and classic bike usage differs across distance, duration, and user characteristics.

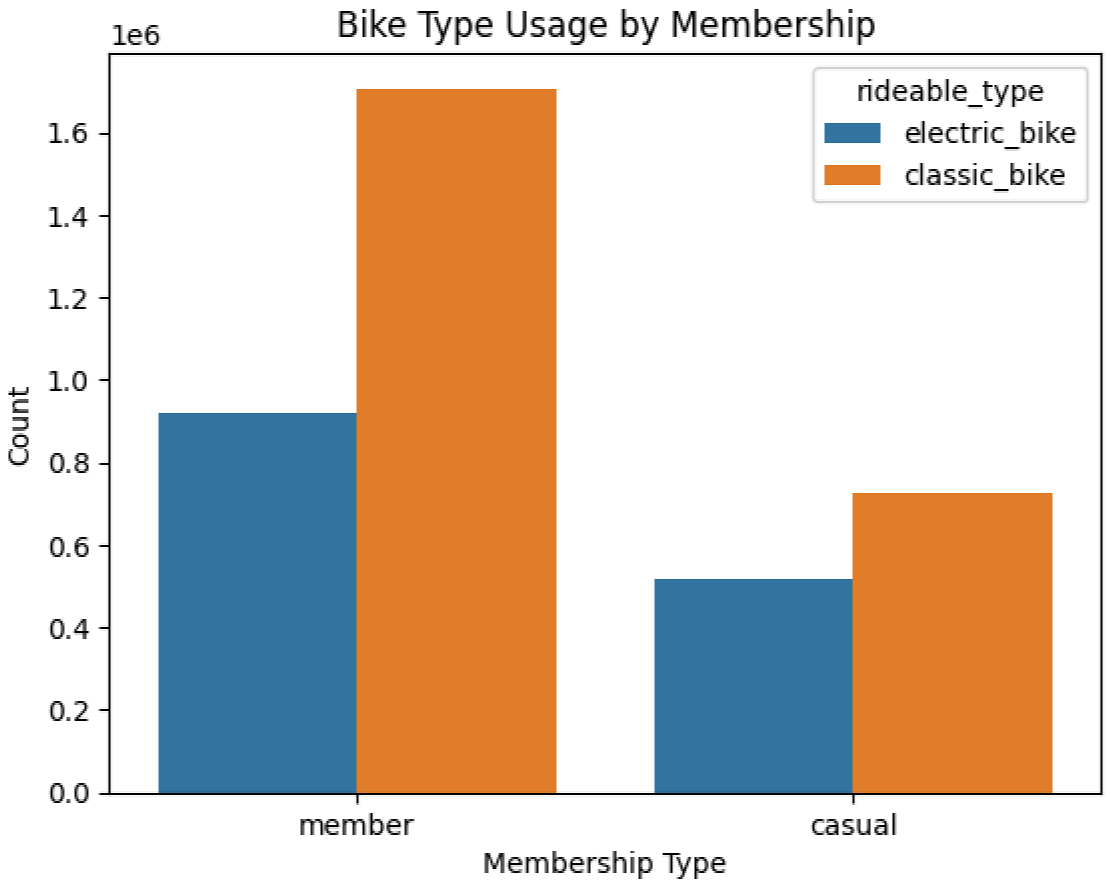
In [43]:
```
# Distance
sns.boxplot(data=df_q3, x='rideable_type', y='distance_km')
plt.title("Distance (km) by Bike Type")
plt.xlabel("Bike Type")
plt.ylabel("Distance (km)")
plt.show()
```



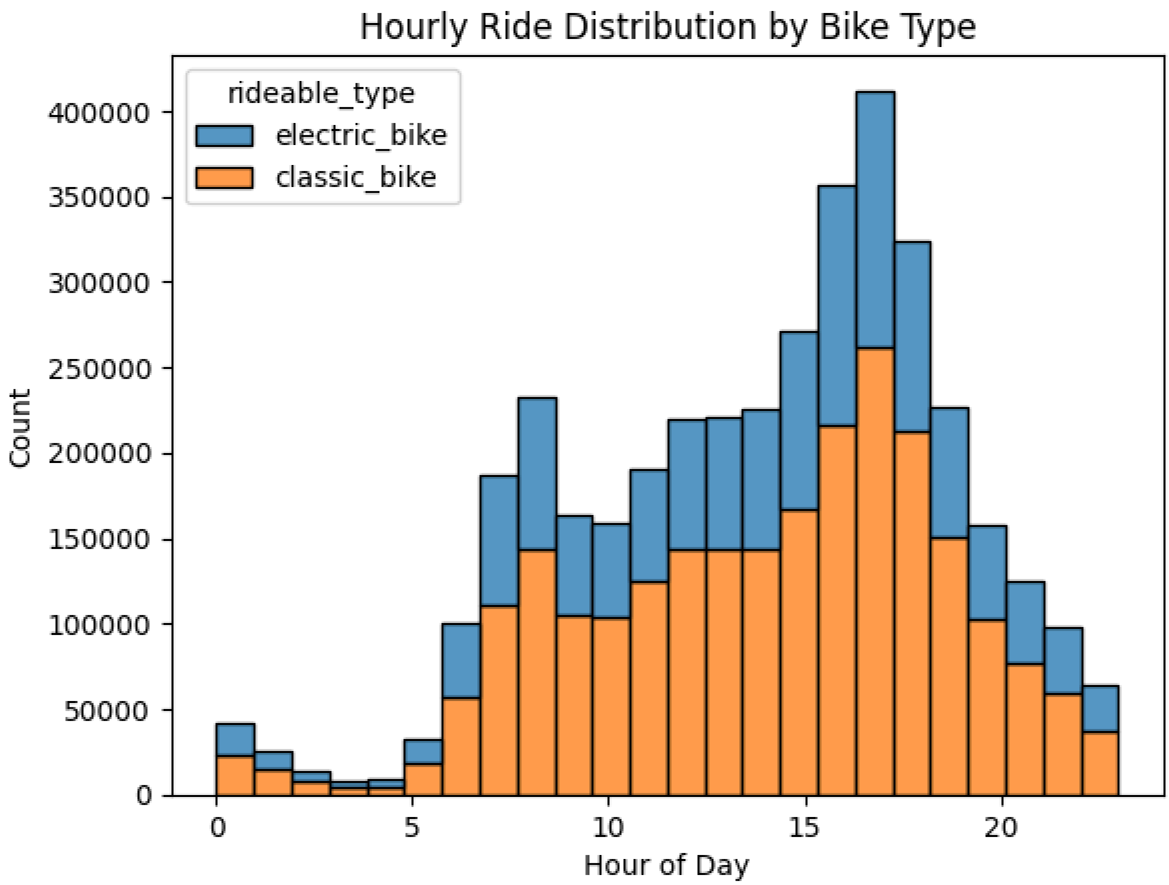
In [44]:
```
# Duration
sns.violinplot(data=df_q3, x='rideable_type', y='duration_min')
plt.title("Duration (min) by Bike Type")
plt.xlabel("Bike Type")
plt.ylabel("Minutes")
plt.show()
```



In [45]:
```
# Membership
sns.countplot(data=df_q3, x='member_casual', hue='rideable_type')
plt.title("Bike Type Usage by Membership")
plt.xlabel("Membership Type")
plt.ylabel("Count")
plt.show()
```



In [46]:
```
# Hourly Distribution
sns.histplot(data=df_q3, x='hour', hue='rideable_type', multiple='stack', bi
plt.title("Hourly Ride Distribution by Bike Type")
plt.xlabel("Hour of Day")
plt.ylabel("Count")
plt.show()
```



Logistic Regression

We now model the probability that a ride uses an electric bike using distance, duration, speed, membership, hour, day-of-week, and season.

In [48]:
```
df_q3.head()
```

Out[48]:

	ride_id	rideable_type	started_at	ended_at	start_station_name	star
0	F96D5A74A3E41399	electric_bike	2023-01-21 20:05:42	2023-01-21 20:16:33	Lincoln Ave & Fullerton Ave	TA
1	13CB7EB698CEDB88	classic_bike	2023-01-10 15:37:36	2023-01-10 15:46:05	Kimbark Ave & 53rd St	TA
2	BD88A2E670661CE5	electric_bike	2023-01-02 07:51:57	2023-01-02 08:05:11	Western Ave & Lunt Ave	
3	C90792D034FED968	classic_bike	2023-01-22 10:52:58	2023-01-22 11:01:44	Kimbark Ave & 53rd St	TA
4	3397017529188E8A	classic_bike	2023-01-12 13:58:01	2023-01-12 14:13:20	Kimbark Ave & 53rd St	TA

5 rows × 25 columns

In [ ]:

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

features = [
    'distance_km', 'duration_min', 'velocity_kmh', 'hour', 'day_of_week', 'i
]

# One-hot encode season and drop first to avoid multicollinearity
X = pd.get_dummies(df_q3[features], drop_first=True)
y = df_q3['electric']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=42
)

model = LogisticRegression(max_iter=3000)
model.fit(X_train, y_train)

model.coef_
```

Bootstrap Resampling

To estimate uncertainty in coefficient estimates, we repeatedly resample our data and refit the logistic regression model.

In [ ]:

```
boot_coefs = []

for i in range(300):
    sample = df_q3.sample(frac=1, replace=True)
    Xb = pd.get_dummies(sample[features], drop_first=True)
    yb = sample['electric']
```

```
    m = LogisticRegression(max_iter=3000)
    m.fit(Xb, yb)

    boot_coefs.append(m.coef_[0])

coef_df = pd.DataFrame(boot_coefs, columns=X.columns)
coef_df.head()
```

In [ ]:
```
# Plot bootstrap distribution
plt.figure(figsize=(12,6))
sns.boxplot(data=coef_df)
plt.xticks(rotation=45, ha='right')
plt.title("Bootstrap Coefficient Distributions")
plt.ylabel("Coefficient Value")
plt.show()
```

In [ ]:
```
from sklearn.metrics import accuracy_score, roc_auc_score, confusion_matrix,

preds = model.predict(X_test)
probs = model.predict_proba(X_test)[:,1]

print("Accuracy:", accuracy_score(y_test, preds))
print("AUC:", roc_auc_score(y_test, probs))
print("Confusion Matrix:\n", confusion_matrix(y_test, preds))

RocCurveDisplay.from_estimator(model, X_test, y_test)
plt.title("ROC Curve")
plt.show()
```

Interpretation and Results

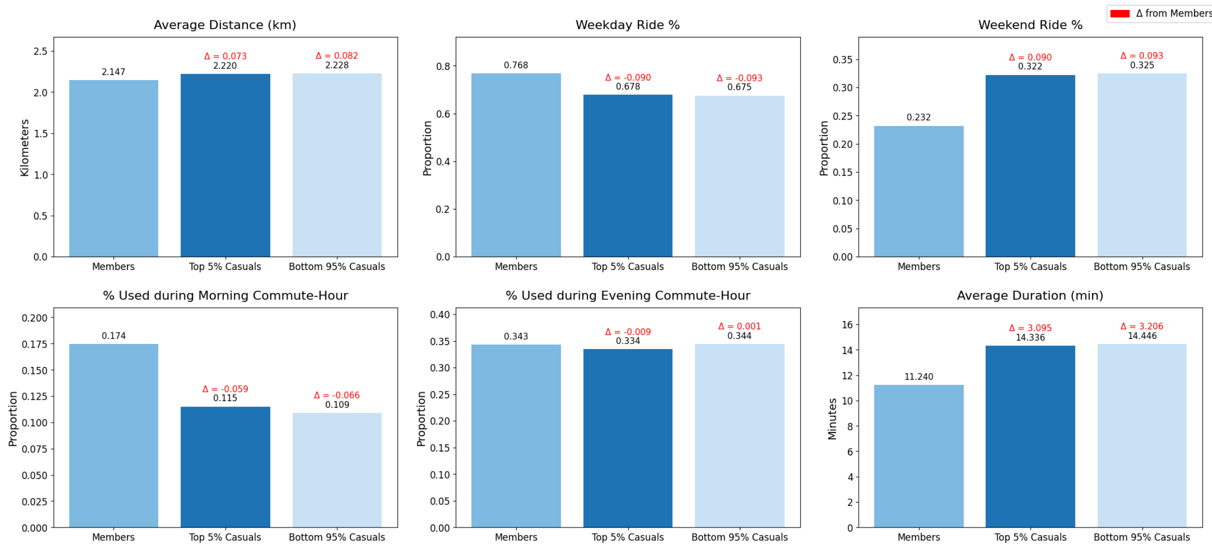
Question 1

Regarding Question 1, once we have the 95% percentile of casual members, we then perform an exploratory analysis to determine whether they exihibit similar behaviors to the Divvy members.

In [51]:
```
from matplotlib.patches import Patch

members_df = ml_divvy_df[ml_divvy_df["member_casual"] == "member"]

plt.rcParams.update({
    "font.size": 12,
    "axes.titlesize": 16,
    "axes.labelsize": 14,
    "xtick.labelsize": 12,
    "ytick.labelsize": 12,
    "legend.fontsize": 12
})
groups = ["Members", "Top 5% Casuals", "Bottom 95% Casuals"]

distance_vals = [
    members_df["distance_km"].mean(),
    top_candidates_df["distance_km"].mean(),
```

```python
    bot_candidates_df["distance_km"].mean()
]

wkday_vals = [
    (members_df["weekday"] <= 4).mean(),
    (top_candidates_df["weekday"] <= 4).mean(),
    (bot_candidates_df["weekday"] <= 4).mean()
]

wkend_vals = [
    (members_df["weekday"] > 4).mean(),
    (top_candidates_df["weekday"] > 4).mean(),
    (bot_candidates_df["weekday"] > 4).mean()
]

TO_COMMUTE_HOURS = [7,8,9]
FROM_COMMUTE_HOURS = [16,17,18,19]

to_commute_vals = [
    members_df["started_at"].dt.hour.isin(TO_COMMUTE_HOURS).mean(),
    top_candidates_df["started_at"].dt.hour.isin(TO_COMMUTE_HOURS).mean(),
    bot_candidates_df["started_at"].dt.hour.isin(TO_COMMUTE_HOURS).mean()
]

from_commute_vals = [
    members_df["started_at"].dt.hour.isin(FROM_COMMUTE_HOURS).mean(),
    top_candidates_df["started_at"].dt.hour.isin(FROM_COMMUTE_HOURS).mean(),
    bot_candidates_df["started_at"].dt.hour.isin(FROM_COMMUTE_HOURS).mean()
]

duration_vals = [
    members_df["ride_duration_min"].mean(),
    top_candidates_df["ride_duration_min"].mean(),
    bot_candidates_df["ride_duration_min"].mean()
]

all_metrics = {
    "Average Distance (km)": (distance_vals, "Kilometers"),
    "Weekday Ride %": (wkday_vals, "Proportion"),
    "Weekend Ride %": (wkend_vals, "Proportion"),
    "% Used during Morning Commute-Hour": (to_commute_vals, "Proportion"),
    "% Used during Evening Commute-Hour": (from_commute_vals, "Proportion"),
    "Average Duration (min)": (duration_vals, "Minutes")
}

fig, axes = plt.subplots(2, 3, figsize=(22, 10))
axes = axes.flatten()

colors = ["#7fbde3", "#1f77b4", "#c9e2f5"]

for ax, (title, (values, ylabel)) in zip(axes, all_metrics.items()):

    bars = ax.bar(groups, values, color=colors)

    ymax = max(values)
    ax.set_ylim(0, ymax * 1.20)
```

```
    # Labels on bars
    for bar in bars:
        height = bar.get_height()
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            height + ymax * 0.02,
            f"{height:.3f}",
            ha='center',
            va='bottom',
            fontsize=11
        )

    # Δ annotations (also lowered)
    member_val = values[0]
    for i in [1, 2]:
        diff = values[i] - member_val
        bar = bars[i]
        ax.text(
            bar.get_x() + bar.get_width() / 2,
            bar.get_height() + ymax * 0.08,
            f"Δ = {diff:.3f}",
            ha='center',
            color="red",
            fontsize=11
        )

    ax.set_title(title, pad=10)
    ax.set_ylabel(ylabel)

# Remove unused subplot if only 5 metrics
if len(all_metrics) < 6:
    fig.delaxes(axes[-1])

delta_patch = Patch(color="red", label="Δ from Members")

fig.legend(
    handles=[delta_patch],
    loc="upper right",
    ncol=3,
    fontsize=12
)

plt.tight_layout(pad=2.0)
plt.show()
```



Once we've obtained the top 5% most likely casual users that exhibit similar behavior to members, we then introduce 6 metrics in order to compare members to the top 5% casual users along with the bottom 95%.

Based on the results of the graph, we notice that the magnitude difference between the top 5% casuals most similar to members and the members is smaller than the difference between the bottom 95% of casual users and members. Even for the "% Used during Commute-Hour" graph, we see that there is virtually a miniscule difference between the top 5% and bottom 95% casual Divvy users.

Delving deeper into each graph:

1. **Average Distance (km)**
   - Members and the top casual users ride nearly identical distances
   - The bottom casual users actually ride longer distances, consistent with tourism
2. **Weekday ride %**
   - Members ride significantly more on the weekdays, consistent with a commuting schedule
   - However, both casual groups ride significantly less on weekdays, meaning there isn't as much of an indicator on its own
3. **Weekend ride %**
   - Similar results and analysis to the weekday ride %
   - Less members ride on the weekend, common with a commute schedule
4. **% Used during Morning Commute Hours (7-9 AM)**
   - We notice an increase in casual users that use Divvy to commute to work, moving closer to the higher percentage of members that use it during those hours
5. **% Used during Evening Commute-Hour (4-7 PM)**
   - The higher proportion of Divvy users during this time could account for a variety of features such as:
     - Dinner plans

        - Commuting home from work
        - Tourism occurring more frequently during the evening
    - Which also aligns with the fact that the more casual users (bottom 95% casuals) use the Divvy bikes during this time
6. **Average Duration (min)**
    - We actually see that the members have a shorter ride time on average, compared to the casual users
    - This could be due to the commuter members having more information on the route they must take, compared to casual users which may use Divvy to ride around the city.

Question 2

With regards to question 2, we resolve to do an exploratory data analysis to uncover trends behind seasonal, temporal, and spatial patterns within the Divvy ridership dataset. This enables us to better determine where to optimally arrange station positioning and the bike volume per station most to Divvy members' and casual riders' benefit.

We separate the analyses into four different containers with subgraphs as follows:

1. **Monthly and Seasonal Ridership Volume** Monthly and seasonal ridership volume serves to detail a less nuanced, more "big picture" type of overview of the number of rides taken, drilled down by season and by month respectively.

    - **Total Rides per Month**
      The graph of total rides per month shows us that ridership volume is strongly proportional to the month, with a steady positive climb from January to August, and then a steep decline from August to December. August is our peak volume month, with around 520k total rides, while December, January, and February are our lowest volume months, with around 130k rides per month.

    - **Total Rides per Season**
      The data we see here is another, less granular form of the previous total rides per month graph. Naturally, the summer season clearly dominates above winter, spring, and fall, with nearly 3x the ridership volume of winter.

    Given these results, the implications or suggestions would be to redistribute bikes to high-stress stations (uncovered in analyses 3 and 4) during the late spring and summer months to handle increased ridership load. On the other hand, a low volume of riders during winter months can justify long-term storage of bikes for this period to avoid idle inventory and unnecessary weathering of utility.

2. **Hourly and Daily Ridership Volume** Hourly and daily ridership volume gives us a more granular look into when exactly riders are typically using Divvy bikes.

- **Hourly Ridership Patterns by Season** We see much of the same results here with regards to the seasons, with summer dominating and winter being the lowest volume. 8:00 AM and 5:00 PM seems to be the peak hours of travel, while midnight hours like 12:00 AM to 4 AM show little to no activity. This makes sense given that commuters traveling to work typically would peak at 8 AM and 5 PM. Still, all seasons show a similar curve.

- **Ridership per Day of Week by Season** Once again, the seasonal volume partitions stay the same. There is a weak upwards trend in summer as the week passes on from the weekdays to the weekends, with Thursday, Friday, and Saturday getting the most volume. On the otherhand, this trend does not hold for spring and winter, with the weekdays getting the most action likely due to commuters.

We can see from the data that peak-hour bike shortages will occur in summer and fall. If we were to do daily dynamic bike rebalancing, we should target the 4–6 PM window, especially in June through September. Summer weekends ought to be prioritized, while spring and winter weekdays should be allocated more volume due to commuters.

3. **Station Statistics by Season** Next, we want to explore the most popular stations to see which stations should be prioritized in case of rebalancing.

   - **Top 15 Stations Per Season By Ride Count** Certain stations like Streeter Dr & Grand Ave and DuSable Lake Shore Dr & North Blvd explode in summer, with ~20k rides each. On the other hand, many stations show balanced spring, winter, and fall demand (Clinton, Kingsbury, Michigan/Oak). Winter usage is pretty scant across the board, while summer usage definitely peaks.

   - **Stations with Highest Seasonal Variability in Ridership Volume** Stations like Montrose Harbor amd DuSable Lake Shore Dr & North Blvd show the highest volatility. They have near-zero winter activity but enormous summer peaks. Most of the other stations with high variability show little activity throughout winter, spring, and summer, with very little peaks in fall.

   In summer, we should expect extreme load on Streeter Dr and Lake Shore Dr nodes, which could use more bike volume at those stations. Contrarily, in winter, bikes can be relocated away from seasonal hotspots to higher baseline commuter stations or maintenance hubs.

4. **Ridership Geospatial Distribution by Season** The geospatial map of latitude and longitude locations by both start and end locations broken down by season shows us that, across all seasons, riders are typically clustered around the same locations. This is great for Divvy because it means that there is no need to reevaluate which geographical locations to prioritize as the seasons change. Typically, we have hubs

of activity within the northeast side of the region around -87.65 longitude and 41.0 latitude.

Question 3

Our analysis reveals several clear patterns in how riders choose between electric and classic bikes. The logistic regression results, supported by bootstrap resampling, show that a combination of trip characteristics, rider attributes, and timing factors all play meaningful roles.

- **Distance, Duration, and Speed:** Electric bikes are used more often for longer and faster trips. The strong positive coefficients for distance and velocity suggest that riders turn to electric bikes when they expect a trip to require more effort or when they want to travel quickly. This aligns with the idea that electric bikes reduce physical strain and shorten travel time, making them especially appealing for extended or time-sensitive rides.

- **Membership Differences:** Casual riders show a significantly higher preference for electric bikes compared to members. While casual riders often prioritize convenience, members tend to be more price-sensitive and may choose classic bikes to minimize costs. This distinction highlights opportunities for targeted promotions—for example, offering discounted electric bike rates to members during certain periods.

- **Time-of-Day and Seasonal Patterns:** Electric bike usage increases during commuting hours and in warmer seasons. Peak-hour usage reflects riders choosing electric bikes to reduce commute time and effort. Higher seasonal usage in summer and fall suggests that electric bikes are also popular for recreational or leisure-oriented rides.

- **Stability of Effects:** Bootstrap resampling confirms that the strongest predictors—distance, velocity, membership status, and peak hours—are highly stable across repeated samples. This increases confidence that these effects reflect real behavioral patterns rather than random variation.

Overall Insight

Together, these results indicate that electric bike choice is driven by a mix of practical needs (speed, convenience), rider characteristics, and temporal factors. These insights can help Divvy optimize station stocking, adjust electric bike availability during high-demand times, and design pricing strategies that better align with rider behavior.

Member Contribution

Each of us created our own avenue to explore, with Leon, Ashley, and Melanie completing each of the three respective questions separately.