In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV,KFold
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder, StandardScaler, FunctionTransformer
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LinearRegression, Lasso, LogisticRegression
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import classification_report, accuracy_score
import pickle
import warnings
from sklearn.metrics import r2_score
from sklearn.model_selection import cross_val_score
import scipy.stats as stats
```

In [2]:
```
warnings.filterwarnings('ignore')
```

In [3]:
```
def Day_of_week(X):
    return pd.to_datetime(X['Date']).dt.day_name().values.reshape(-1,1)

def Timeslot(X):
    def time_slot(time_str):
        hour = int(time_str.split(':')[0])
        if 10 <= hour < 12:
            return 'Morning'
        elif 12 <= hour < 17:
            return 'Afternoon'
        elif 17 <= hour < 19:
            return 'Evening'
        else:
            return 'Night'

    return pd.Series(X['Time']).apply(time_slot).values.reshape(-1,1)
```

Import models, X_test and t_test files for all models

In [4]:
```
# Load X_test and y_test using pickle

with open('X1_test.pkl', 'rb') as f:
    X1_test= pickle.load(f)

with open('t1_test.pkl', 'rb') as f:
    t1_test= pickle.load(f)
```

```python
with open('X2_test.pkl', 'rb') as f:
    X2_test= pickle.load(f)

with open('t2_test.pkl', 'rb') as f:
    t2_test= pickle.load(f)

with open('X3_test.pkl', 'rb') as f:
    X3_test= pickle.load(f)

with open('t3_test.pkl', 'rb') as f:
    t3_test= pickle.load(f)

with open('X4_test.pkl', 'rb') as f:
    X4_test= pickle.load(f)

with open('t4_test.pkl', 'rb') as f:
    t4_test= pickle.load(f)

with open('X5_test.pkl', 'rb') as f:
    X5_test= pickle.load(f)

with open('t5_test.pkl', 'rb') as f:
    t5_test= pickle.load(f)


with open('model1_lr.pkl', 'rb') as f:
    model1_lr = pickle.load(f)
with open('model1_lasso.pkl', 'rb') as f:
    model1_lasso = pickle.load(f)
with open('model2_lr.pkl', 'rb') as f:
    model2_lr = pickle.load(f)
with open('model2_lasso.pkl', 'rb') as f:
    model2_lasso = pickle.load(f)
with open('model3.pkl', 'rb') as f:
    model3 = pickle.load(f)
with open('model4.pkl', 'rb') as f:
    model4 = pickle.load(f)
with open('model5.pkl', 'rb') as f:
    model5 = pickle.load(f)
with open('model6.pkl', 'rb') as f:
    model6 = pickle.load(f)
```

R2score with its confidence intervals for Multiple linear regression with and without Lasso for gross income and Unit price respectively

In [5]:

```python
y1_pred_lr = model1_lr.predict(X1_test)
y1_pred_lasso = model1_lasso.predict(X1_test)
y2_pred_lr = model2_lr.predict(X2_test)
```

```
y2_pred_lasso = model2_lasso.predict(X2_test)
y3_pred = model3.predict(X3_test)
y4_pred = model4.predict(X4_test)
```

In [6]:

```
def evaluate(X_test,t_test,y_pred, model,model_name):
    ci=0.95
    r2 = r2_score(t_test, y_pred)
    print(f"model name:",model_name)
    print(f"R² on test set: {r2}")

    n = len(t_test)  # number of observations
    k = X_test.shape[1]  # number of predictors
    se_r2 = np.sqrt((1 - r2**2) / (n - k - 1))
    alpha = 1 - ci
    z_score = stats.norm.ppf(1 - alpha / 2)
    margin_of_error = z_score * se_r2

    lower_bound = r2 - margin_of_error
    upper_bound = r2 + margin_of_error
    print("95% confidence interval:",[lower_bound,upper_bound])
```

In [7]:

```
evaluate(X1_test,t1_test,y1_pred_lr, model1_lr, 'Linear regression to predict gross income')
print('\n')
evaluate(X1_test,t1_test,y1_pred_lasso, model1_lasso, 'Linear regression with lasso to predict gross income')
print('\n')
evaluate(X2_test,t2_test,y2_pred_lr, model2_lr, 'Linear regression to predict Unit price')
print('\n')
evaluate(X2_test,t2_test,y2_pred_lasso, model2_lasso, 'Linear regression with lasso to predict Unit price')
print('\n')
```

```
model name: Linear regression to predict gross income
R² on test set: 0.8911494702615879
95% confidence interval: [0.8264656904440684, 0.9558332500791074]


model name: Linear regression with lasso to predict gross income
R² on test set: 0.8905189653373747
95% confidence interval: [0.8256589339164472, 0.9553789967583023]


model name: Linear regression to predict Unit price
R² on test set: 0.7777872241367431
95% confidence interval: [0.6877023885046276, 0.8678720597688585]


model name: Linear regression with lasso to predict Unit price
R² on test set: 0.7868333214220291
95% confidence interval: [0.6983769809306214, 0.8752896619134368]
```

Accuracy and its confidence intervals for classifying day of purchase

In [8]:
```
y5_pred = model5.predict(X5_test)
y6_pred = model6.predict(X5_test)
```

In [9]:
```
def accurate(X_test,t_test,y_pred, model,model_name):
    ci=0.95
    accuracy = accuracy_score(t_test, y_pred)
    print(f"model name:",model_name)
    print(f"Accuracy on test set: {accuracy}")

    n = len(t_test)  # number of observations
    se_acc = np.sqrt((accuracy*(1-accuracy))/n)
    alpha = 1 - ci
    z_score = stats.norm.ppf(1 - alpha / 2)
    margin_of_error = z_score * se_acc

    lower_bound = max(0, accuracy - margin_of_error)
    upper_bound = min(1, accuracy + margin_of_error)
    print("95% confidence interval:",[lower_bound,upper_bound])
```

In [10]:
```
accurate(X5_test,t5_test,y5_pred, model5, "Decision tree classifier to classify day of purchase")
accurate(X5_test,t5_test,y6_pred, model6, "Random Forest classifier to classify day of purchase")
```

```
model name: Decision tree classifier to classify day of purchase
Accuracy on test set: 0.16
95% confidence interval: [0.10919192655191055, 0.21080807344808944]
model name: Random Forest classifier to classify day of purchase
Accuracy on test set: 0.15
95% confidence interval: [0.10051333514781477, 0.19948666485218522]
```

In [ ]: