Project

December 6, 2023

[1]:
```python
from __future__ import print_function
from packaging.version import parse as Version
from platform import python_version

OK = '\x1b[42m[ OK ]\x1b[0m'
FAIL = "\x1b[41m[FAIL]\x1b[0m"

try:
    import importlib
except ImportError:
    print(FAIL, "Python version 3.11 is required,"
                " but %s is installed." % sys.version)

def import_version(pkg, min_ver, fail_msg=""):
    mod = None
    try:
        mod = importlib.import_module(pkg)
        if pkg in {'PIL'}:
            ver = mod.VERSION
        else:
            ver = mod.__version__
        if Version(ver) == Version(min_ver):
            print(OK, "%s version %s is installed."
                  % (lib, min_ver))
        else:
            print(FAIL, "%s version %s is required, but %s installed."
                  % (lib, min_ver, ver))
    except ImportError:
        print(FAIL, '%s not installed. %s' % (pkg, fail_msg))
    return mod


# first check the python version
pyversion = Version(python_version())

if pyversion >= Version("3.11.4"):
    print(OK, "Python version is %s" % pyversion)
```

```
elif pyversion < Version("3.11"):
    print(FAIL, "Python version 3.11 is required,"
                " but %s is installed." % pyversion)
else:
    print(FAIL, "Unknown Python version: %s" % pyversion)


print()
requirements = {'numpy': "1.24.4", 'matplotlib': "3.7.2",'sklearn': "1.3.0",
                'pandas': "2.0.3",'xgboost': "1.7.6", 'shap': "0.42.1",␣
 ↪'seaborn': "0.12.2"}

# now the dependencies
for lib, required_version in list(requirements.items()):
    import_version(lib, required_version)
```

```
[ OK ] Python version is 3.11.4

[ OK ] numpy version 1.24.4 is installed.
[ OK ] matplotlib version 3.7.2 is installed.
[ OK ] sklearn version 1.3.0 is installed.
[ OK ] pandas version 2.0.3 is installed.
[ OK ] xgboost version 1.7.6 is installed.
[ OK ] shap version 0.42.1 is installed.
[ OK ] seaborn version 0.12.2 is installed.

Using `tqdm.autonotebook.tqdm` in notebook mode. Use `tqdm.tqdm` instead to
force console mode (e.g. in jupyter console)
```

[2]:
```
# load packages
import pandas as pd
import numpy as np
import matplotlib
from matplotlib import pylab as plt
import seaborn as sns

# import data
df = pd.read_csv('/Users/yingyizhu/Desktop/DATA1030/DATA1030-Fall2023/Midterm␣
 ↪Project/london_merged.csv')
df.describe().T
df
```

[2]:

	timestamp	cnt	t1	t2	hum	wind_speed	weather_code	\
0	2015-01-04 00:00:00	182	3.0	2.0	93.0	6.0	3.0	
1	2015-01-04 01:00:00	138	3.0	2.5	93.0	5.0	1.0	
2	2015-01-04 02:00:00	134	2.5	2.5	96.5	0.0	1.0	
3	2015-01-04 03:00:00	72	2.0	2.0	100.0	0.0	1.0	
4	2015-01-04 04:00:00	47	2.0	0.0	93.0	6.5	1.0	

```
…                        …    …   …   …      …           …              …
17409  2017-01-03 19:00:00  1042  5.0  1.0   81.0        19.0           3.0
17410  2017-01-03 20:00:00   541  5.0  1.0   81.0        21.0           4.0
17411  2017-01-03 21:00:00   337  5.5  1.5   78.5        24.0           4.0
17412  2017-01-03 22:00:00   224  5.5  1.5   76.0        23.0           4.0
17413  2017-01-03 23:00:00   139  5.0  1.0   76.0        22.0           2.0

       is_holiday  is_weekend  season
0             0.0         1.0     3.0
1             0.0         1.0     3.0
2             0.0         1.0     3.0
3             0.0         1.0     3.0
4             0.0         1.0     3.0
…             …           …       …
17409         0.0         0.0     3.0
17410         0.0         0.0     3.0
17411         0.0         0.0     3.0
17412         0.0         0.0     3.0
17413         0.0         0.0     3.0

[17414 rows x 10 columns]
```

[3]:

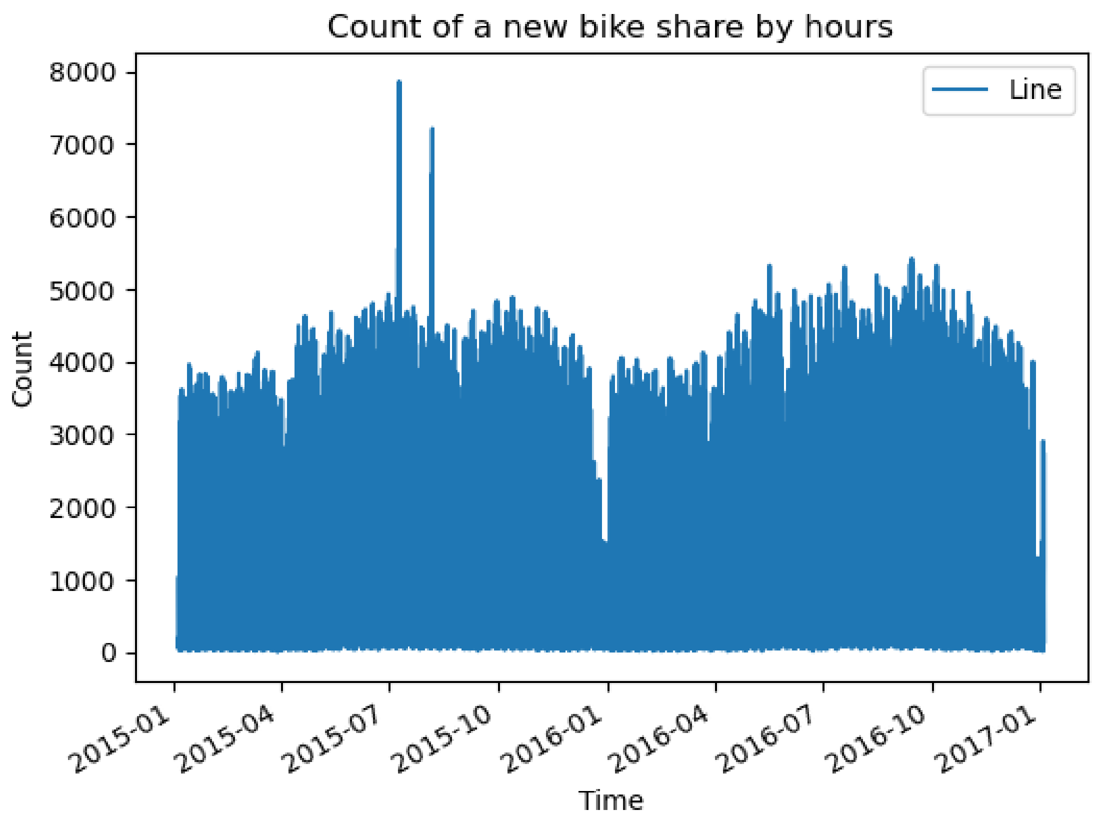
```python
# print data types
print(df.dtypes)
df.info()
```

```
timestamp        object
cnt               int64
t1              float64
t2              float64
hum             float64
wind_speed      float64
weather_code    float64
is_holiday      float64
is_weekend      float64
season          float64
dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17414 entries, 0 to 17413
Data columns (total 10 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   timestamp     17414 non-null  object
 1   cnt           17414 non-null  int64
 2   t1            17414 non-null  float64
 3   t2            17414 non-null  float64
 4   hum           17414 non-null  float64
 5   wind_speed    17414 non-null  float64
```

```
 6   weather_code  17414 non-null  float64
 7   is_holiday    17414 non-null  float64
 8   is_weekend    17414 non-null  float64
 9   season        17414 non-null  float64
dtypes: float64(8), int64(1), object(1)
memory usage: 1.3+ MB
```

[4]:
```
# count
print(df['cnt'].value_counts())
print(df['cnt'].describe())

# Convert 'timestamp' column to datetime
df['timestamp'] = pd.to_datetime(df['timestamp'])

# Create a line plot using 'timestamp' as x-axis and 'cnt' as y-axis
df.plot(x='timestamp', y='cnt', kind='line')

plt.xlabel('Time')
plt.ylabel('Count')
plt.title('Count of a new bike share by hours')
plt.legend(['Line'])

plt.show()
```

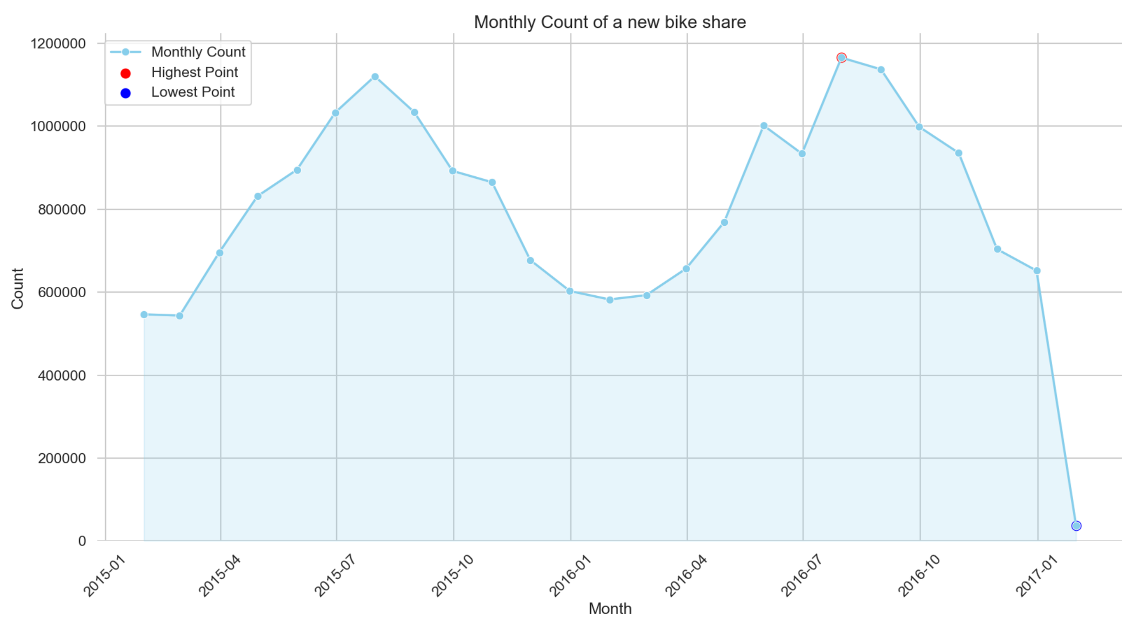
```
cnt
46      46
53      39
33      36
70      36
120     36
        ..
3022     1
3112     1
1338     1
3270     1
2220     1
Name: count, Length: 3781, dtype: int64
count    17414.000000
mean      1143.101642
std       1085.108068
min          0.000000
25%        257.000000
50%        844.000000
75%       1671.750000
max       7860.000000
Name: cnt, dtype: float64
```



[5]:

```python
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker

df['timestamp'] = pd.to_datetime(df['timestamp'])
monthly_count = df.resample('M', on='timestamp').sum()['cnt']
data_plot = monthly_count.reset_index(name='cnt')

sns.set_style("whitegrid")
sns.set_context("talk")

# Choose a color palette
palette = sns.color_palette("coolwarm", n_colors=len(data_plot))
plt.figure(figsize=(18, 10))
sns.lineplot(x='timestamp', y='cnt', data=data_plot, marker='o', linestyle='-',␣
↪color="skyblue", linewidth=2.5, label='Monthly Count')
plt.fill_between(data_plot['timestamp'], data_plot['cnt'], color="skyblue",␣
↪alpha=0.2)
max_idx = data_plot['cnt'].idxmax()
min_idx = data_plot['cnt'].idxmin()
```

```
plt.scatter(data_plot['timestamp'][max_idx], data_plot['cnt'][max_idx],
    color='red', s=100, label='Highest Point')
plt.scatter(data_plot['timestamp'][min_idx], data_plot['cnt'][min_idx],
    color='blue', s=100, label='Lowest Point')
plt.xlabel('Month')
plt.ylabel('Count')
plt.title('Monthly Count of a new bike share', fontsize=20)
plt.gca().get_yaxis().set_major_formatter(ticker.FuncFormatter(lambda x, _:
    int(x)))
plt.ylim(bottom=0)
plt.xticks(rotation=45)
plt.legend(loc='upper left')
sns.despine(left=True, bottom=True)
plt.tight_layout()
plt.show()
```



[6]:
```
# no duplicated rows
df.duplicated().sum()
```

[6]: 0

[7]:
```
# season

df['season'].replace(0, 'Spring', inplace=True)
df['season'].replace(1, 'Summer', inplace=True)
df['season'].replace(2, 'Fall', inplace=True)
df['season'].replace(3, 'Winter', inplace=True)
```

[8]:
```python
#Is Holiday

df['is_holiday'].replace(1, 'Yes', inplace=True)
df['is_holiday'].replace(0, 'No', inplace=True)
```

[9]:
```python
# Is weekend

df['is_weekend'].replace(1, 'Yes', inplace=True)
df['is_weekend'].replace(0, 'No', inplace=True)
```

[10]:
```python
#Weather Codes

df['weather_code'].replace(1, 'Clear', inplace=True)
df['weather_code'].replace(2, 'Scattered clouds', inplace=True)
df['weather_code'].replace(3, 'Broken clouds', inplace=True)
df['weather_code'].replace(4, 'Cloudy', inplace=True)
df['weather_code'].replace(7, 'Rain/Light Rain', inplace=True)
df['weather_code'].replace(10, 'Rain with thunderstorm', inplace=True)
df['weather_code'].replace(26, 'snowfall', inplace=True)
```

[11]:
```python
# check the dataset again
df.head(10)
```

[11]:
```
             timestamp  cnt   t1   t2    hum  wind_speed   weather_code  \
0 2015-01-04 00:00:00  182  3.0  2.0   93.0         6.0  Broken clouds
1 2015-01-04 01:00:00  138  3.0  2.5   93.0         5.0          Clear
2 2015-01-04 02:00:00  134  2.5  2.5   96.5         0.0          Clear
3 2015-01-04 03:00:00   72  2.0  2.0  100.0         0.0          Clear
4 2015-01-04 04:00:00   47  2.0  0.0   93.0         6.5          Clear
5 2015-01-04 05:00:00   46  2.0  2.0   93.0         4.0          Clear
6 2015-01-04 06:00:00   51  1.0 -1.0  100.0         7.0         Cloudy
7 2015-01-04 07:00:00   75  1.0 -1.0  100.0         7.0         Cloudy
8 2015-01-04 08:00:00  131  1.5 -1.0   96.5         8.0         Cloudy
9 2015-01-04 09:00:00  301  2.0 -0.5  100.0         9.0  Broken clouds

  is_holiday is_weekend  season
0         No        Yes  Winter
1         No        Yes  Winter
2         No        Yes  Winter
3         No        Yes  Winter
4         No        Yes  Winter
5         No        Yes  Winter
6         No        Yes  Winter
7         No        Yes  Winter
8         No        Yes  Winter
9         No        Yes  Winter
```

[12]:

```
df['timestamp'].head()
print(df['weather_code'].value_counts())
```

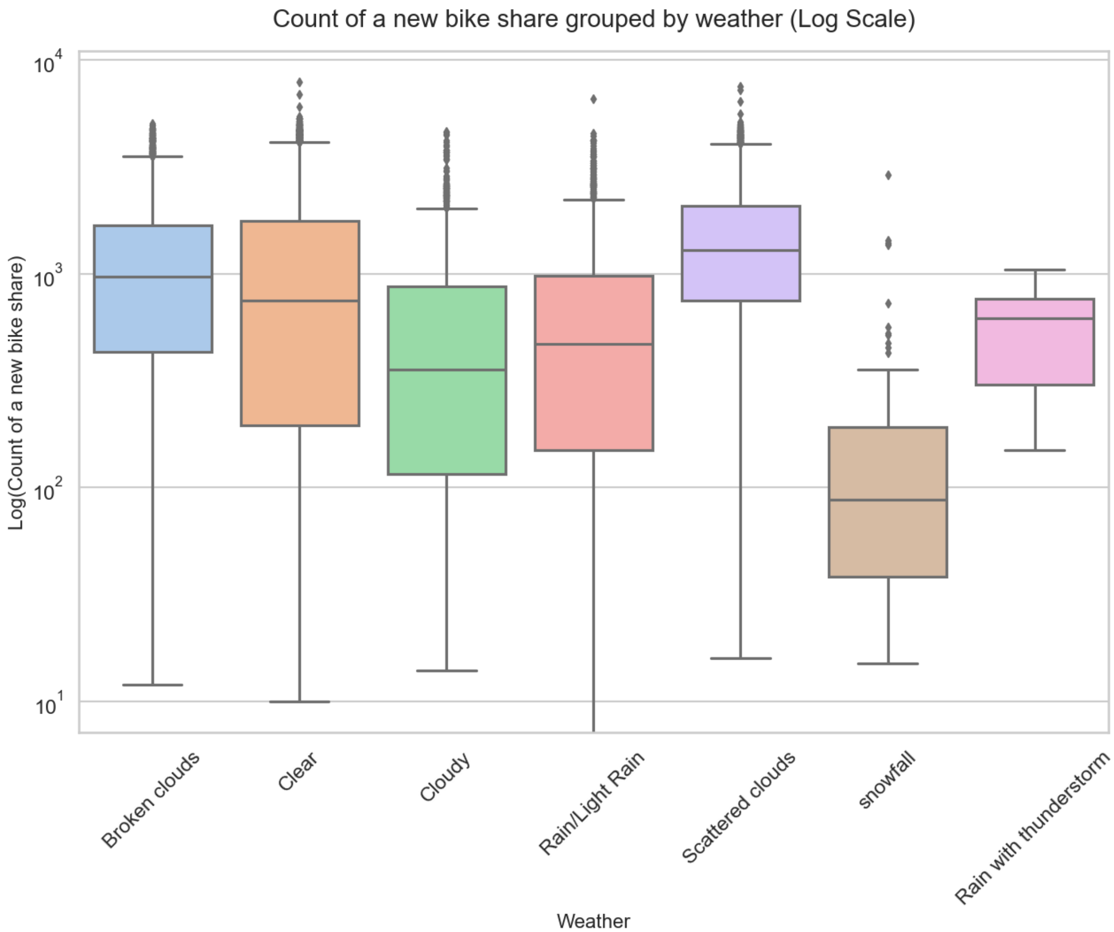
```
weather_code
Clear                      6150
Scattered clouds           4034
Broken clouds              3551
Rain/Light Rain            2141
Cloudy                     1464
snowfall                     60
Rain with thunderstorm       14
Name: count, dtype: int64
```

[13]:

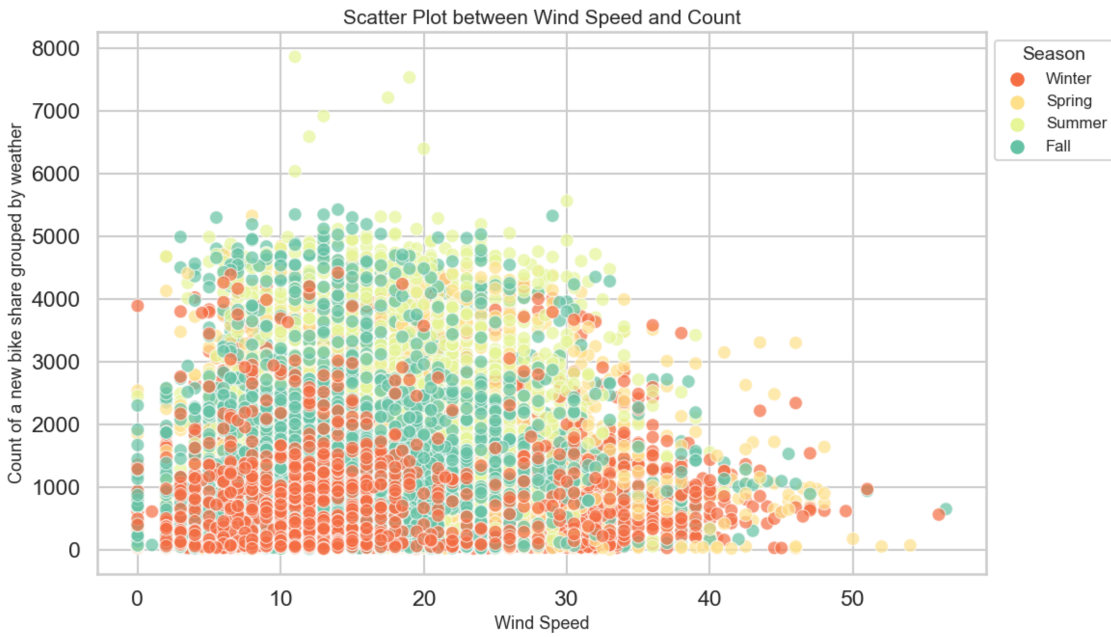
```
sns.set_style("whitegrid")
sns.set_palette("pastel")

fig, ax = plt.subplots(figsize=(15, 10))

sns.boxplot(x='weather_code', y='cnt', data=df, ax=ax)
plt.xticks(rotation=45)

ax.set_yscale('log')

ax.set_xlabel('Weather', fontsize=16)
ax.set_ylabel('Log(Count of a new bike share)', fontsize=16)
ax.set_title('Count of a new bike share grouped by weather (Log Scale)',␣
↪fontsize=20, pad=20)
```

[13]:

```
Text(0.5, 1.0, 'Count of a new bike share grouped by weather (Log Scale)')
```

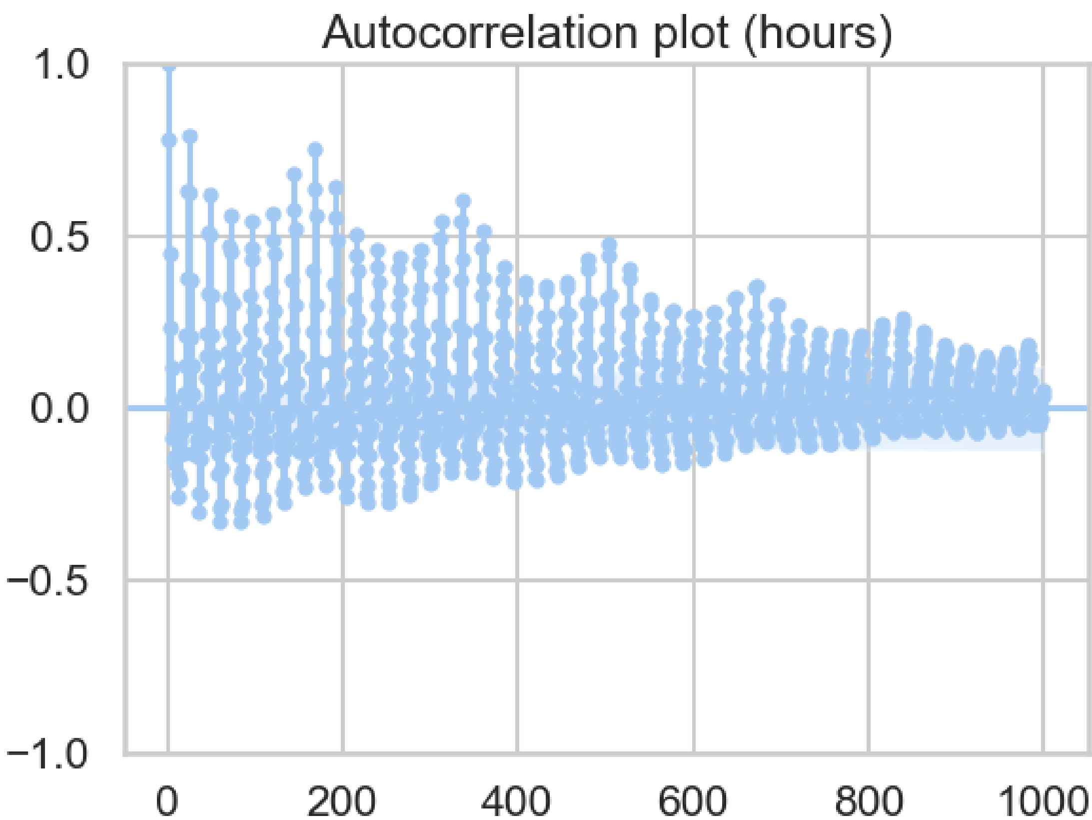


[14]:
```
plt.figure(figsize=(12, 7))
sns.scatterplot(x='wind_speed', y='cnt', hue='season', data=df,␣
  ↪palette='Spectral', s=100, edgecolor='w', alpha=0.7)

plt.xlabel('Wind Speed', fontsize=13)
plt.ylabel('Count of a new bike share grouped by weather', fontsize=13)
plt.title('Scatter Plot between Wind Speed and Count', fontsize=15)
plt.legend(title='Season', title_fontsize='14', fontsize=12, loc='upper left',␣
  ↪bbox_to_anchor=(1,1))
plt.tight_layout()
plt.show()
```

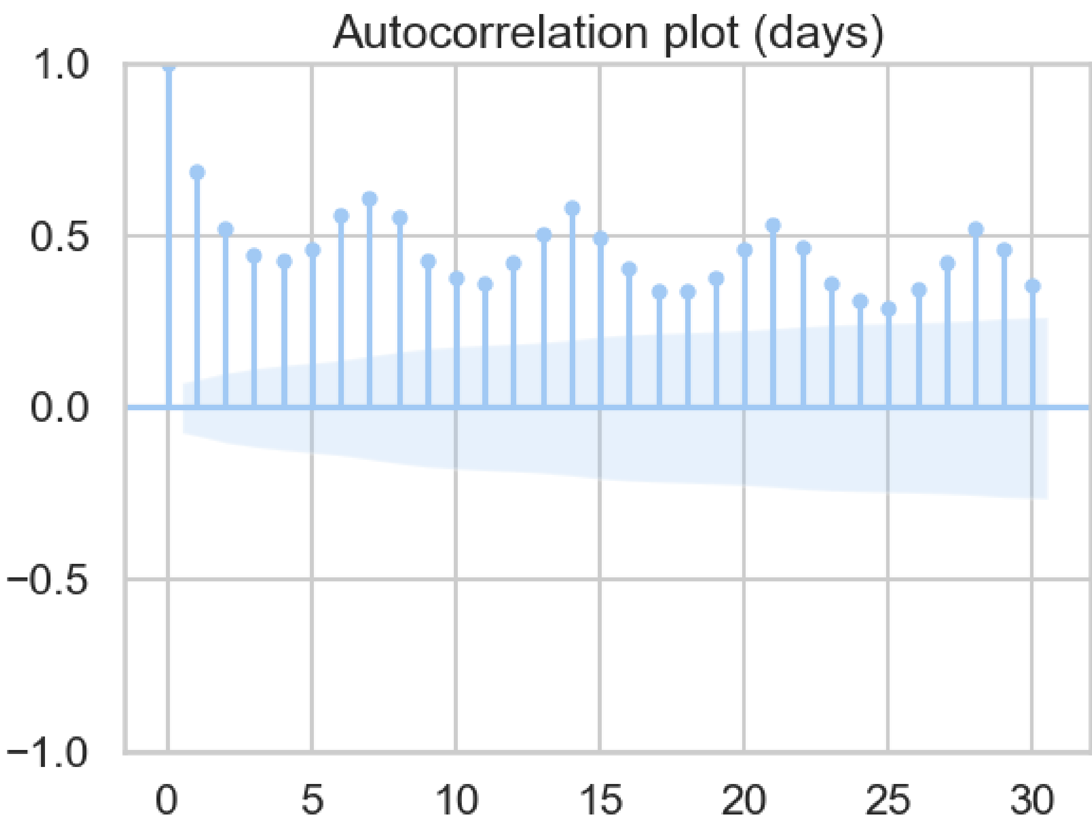


[15]:
```
import statsmodels.api as sm

plt.figure(figsize=(4,4))

sm.graphics.tsa.plot_acf(df['cnt'], lags=1000)
plt.title("Autocorrelation plot (hours)")
plt.tight_layout()
plt.show()

daily_df = df.resample('D', on='timestamp').sum()  # Assuming 'timestamp' is␣
↪the column with datetime data.
plt.figure(figsize=(10,6))
sm.graphics.tsa.plot_acf(daily_df['cnt'], lags=30)  # For example, here it's␣
↪showing 30 days of lags.
plt.title("Autocorrelation plot (days)")
plt.tight_layout()
plt.show()
```

<Figure size 400x400 with 0 Axes>




```
<Figure size 1000x600 with 0 Axes>
```



[16]:

```
import pandas as pd
import numpy as np
from datetime import datetime
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.linear_model import Ridge
from sklearn.neighbors import KNeighborsRegressor
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split, GridSearchCV,␣
↪cross_val_score, TimeSeriesSplit
from sklearn.metrics import mean_squared_log_error
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder, MinMaxScaler
import matplotlib.pyplot as plt
import warnings
from sklearn.dummy import DummyRegressor

warnings.filterwarnings("ignore")
%matplotlib inline
```

```
# Load the dataset
data = pd.read_csv("/Users/yingyizhu/Desktop/DATA1030/DATA1030-Fall2023/Midterm␣
↪Project/london_merged.csv")
```

[17]:
```
# Convert 'timestamp' to datetime and extract features
data['timestamp'] = pd.to_datetime(data['timestamp'])
data['hour'] = data['timestamp'].dt.hour
data['day'] = data['timestamp'].dt.day
data['month'] = data['timestamp'].dt.month
data['year'] = data['timestamp'].dt.year
data['day_of_week'] = data['timestamp'].dt.dayofweek

# Create lagged features
lag_hours = [3, 2, 1]
for lag in lag_hours:
    data[f'lag_{lag}_hour'] = data['cnt'].shift(lag)

# Drop rows with NaN values and the original 'timestamp' column
data.dropna(inplace=True)
data.drop('timestamp', axis=1, inplace=True)
```

[18]:
```
onehot_ftrs = ['weather_code', 'is_holiday', 'is_weekend', 'season']
minmax_ftrs = ['hum']
std_ftrs = ['t1', 't2', 'lag_3_hour', 'lag_2_hour', 'lag_1_hour']

preprocessor = ColumnTransformer(
    transformers=[
        ('onehot', OneHotEncoder(handle_unknown='ignore'), onehot_ftrs),
        ('minmax', MinMaxScaler(), minmax_ftrs),
        ('std', StandardScaler(), std_ftrs)])

clf = Pipeline(steps=[('preprocessor', preprocessor)])

# Train-test split
X = data.drop('cnt', axis=1)
y = np.log1p(data['cnt'])  # Log transformation of 'cnt'
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

# Apply preprocessing
X_train_prep = clf.fit_transform(X_train)
X_test_prep = clf.transform(X_test)
```

[19]:
```
print(X_train_prep)
```

```
[[ 0.          0.          0.         … -0.16636878 -0.31807903
  -0.34518542]
 [ 1.          0.          0.         …  1.24199525  2.8617132
```

```
  2.52070303]
 [ 0.          0.          0.         … -0.9692939  -0.9809583
 -0.99706701]
 …
 [ 1.          0.          0.         … -0.92942579 -0.43608814
  0.89513653]
 [ 0.          1.          0.         … -0.74862857 -0.82791524
 -0.88489557]
 [ 0.          0.          1.         …  0.23231228  0.5337992
  0.5852859 ]]
```

[20]:
```
print(X_test_prep)
```

```
[[ 1.          0.          0.         … -0.98227421 -0.96805106
 -1.00993915]
 [ 0.          0.          0.         … -0.69485298 -0.41027364
 -0.26703319]
 [ 0.          1.          0.         … -0.52518174 -0.69331111
 -0.41138496]
 …
 [ 1.          0.          0.         …  2.51777465  2.13982937
  1.58931227]
 [ 1.          0.          0.         …  0.33337329  0.20650832
  0.38025121]
 [ 0.          0.          1.         …  0.5920524   0.22955698
 -0.21554466]]
```

[21]:
```
def rmsle(y_true, y_pred):
    return np.sqrt(mean_squared_log_error(np.exp(y_true), np.exp(y_pred)))

def tune_and_evaluate_model(model, params, X, y, model_name):
    tscv = TimeSeriesSplit(n_splits=5)
    grid_search = GridSearchCV(model, params, cv=tscv,␣
 ↪scoring='neg_mean_squared_log_error', n_jobs=-1)
    grid_search.fit(X, y)
    best_model = grid_search.best_estimator_
    scores = cross_val_score(best_model, X, y, cv=tscv,␣
 ↪scoring='neg_mean_squared_log_error')
    mean_rmsle = np.mean(np.sqrt(-scores))
    std_rmsle = np.std(np.sqrt(-scores))

    print(f"{model_name} - Best Params: {grid_search.best_params_}")
    print(f"{model_name} - RMSLE: {rmsle(y, best_model.predict(X))}")
    print(f"{model_name} - Mean RMSLE: {mean_rmsle}, Std RMSLE: {std_rmsle}\n")

    return best_model, mean_rmsle, std_rmsle
```

[22]:
```
# Ridge Regression Model
print("Ridge Regression Model")
ridge_params = {'alpha': [0.1, 1, 10, 50, 100, 200]}
ridge_best_model, ridge_mean_rmsle, ridge_std_rmsle = tune_and_evaluate_model(Ridge(), ridge_params, X_train_prep, y_train, "Ridge Regression")
```

```
Ridge Regression Model
Ridge Regression - Best Params: {'alpha': 0.1}
Ridge Regression - RMSLE: 0.777290231095798
Ridge Regression - Mean RMSLE: 0.11724054819499727, Std RMSLE:
0.0007334746285O9329
```

[23]:
```
print("Random Forest Model")
rf_params = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None]
}
rf_best_model, rf_mean_rmsle, rf_std_rmsle = tune_and_evaluate_model(RandomForestRegressor(random_state=42), rf_params, X_train_prep, y_train, "Random Forest")
```

```
Random Forest Model
Random Forest - Best Params: {'max_depth': 20, 'n_estimators': 200}
Random Forest - RMSLE: 0.08727214801637538
Random Forest - Mean RMSLE: 0.039257593135102076, Std RMSLE:
0.001755653430975494
```

[24]:
```
print("Support Vector Regression Model")
svr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('svr', SVR())
])
svr_params = {
    'svr__C': [0.1, 1, 10],
    'svr__gamma': ['scale', 'auto']
}
svr_best_model, svr_mean_rmsle, svr_std_rmsle = tune_and_evaluate_model(svr_pipeline, svr_params, X_train_prep, y_train, "Support Vector Regression")
```

```
Support Vector Regression Model
Support Vector Regression - Best Params: {'svr__C': 10, 'svr__gamma': 'scale'}
Support Vector Regression - RMSLE: 0.39223715682574706
Support Vector Regression - Mean RMSLE: 0.07024241270436024, Std RMSLE:
0.002875009062252405
```

[25]:
```
print("K-Neighbors Regressor Model")
knn_params = {
    'n_neighbors': [3, 5, 7],
    'weights': ['uniform', 'distance']
}
knn_best_model, knn_mean_rmsle, knn_std_rmsle =␣
 ↪tune_and_evaluate_model(KNeighborsRegressor(), knn_params, X_train_prep,␣
 ↪y_train, "K-Neighbors Regressor")
```

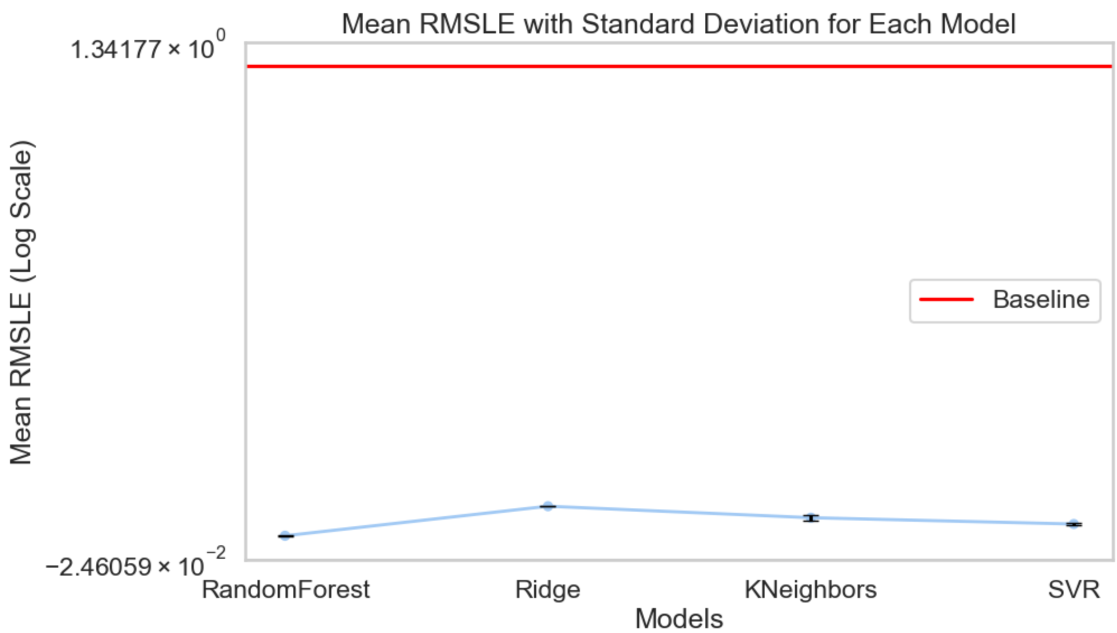
```
K-Neighbors Regressor Model
K-Neighbors Regressor - Best Params: {'n_neighbors': 5, 'weights': 'distance'}
K-Neighbors Regressor - RMSLE: 0.0
K-Neighbors Regressor - Mean RMSLE: 0.0869649993926992, Std RMSLE:
0.00731023548406084
```

[26]:
```
print("Baseline Model")
baseline_model = DummyRegressor(strategy="mean")
baseline_rmsle = rmsle(y_train, baseline_model.fit(X_train_prep, y_train).
 ↪predict(X_train_prep))
print(f"Baseline Model - RMSLE: {baseline_rmsle}")
```

```
Baseline Model
Baseline Model - RMSLE: 1.2796591225393974
```

[27]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Model names
models = ['RandomForest', 'Ridge', 'KNeighbors', 'SVR']

# Mean RMSLE values for each model (Replace these with your actual values)
mean_rmsles = [rf_mean_rmsle, ridge_mean_rmsle, knn_mean_rmsle, svr_mean_rmsle]

# Standard deviation of RMSLE for each model (Replace these with your actual␣
 ↪values)
std_rmsles = [rf_std_rmsle, ridge_std_rmsle, knn_std_rmsle, svr_std_rmsle]

# Baseline RMSLE (Replace with your actual value)
baseline_rmsle = baseline_rmsle

# Plotting
plt.figure(figsize=(10, 6))
sns.pointplot(x=models, y=mean_rmsles, scale=0.5)
plt.errorbar(x=models, y=mean_rmsles, yerr=std_rmsles, fmt='none', capsize=5,␣
 ↪color='black')
plt.axhline(y=baseline_rmsle, color='r', linestyle='-', label='Baseline')
plt.yscale('symlog')  # Log scale for better visibility
```

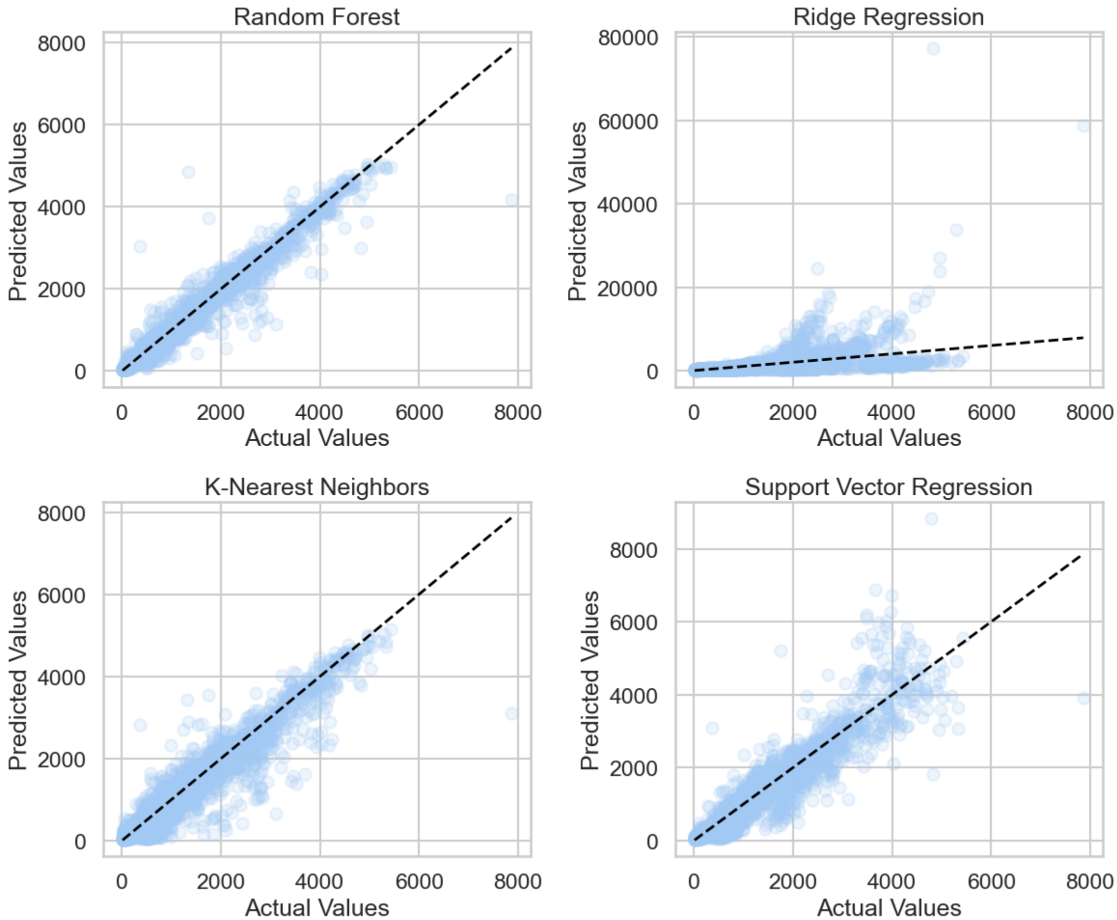
```
plt.xlabel('Models')
plt.ylabel('Mean RMSLE (Log Scale)')
plt.title('Mean RMSLE with Standard Deviation for Each Model')
plt.legend()

# Save the plot
plt.savefig("mean_rmsle_with_std.png")
plt.show()
```



[29]:

```
models = [
    ('Random Forest', rf_best_model),
    ('Ridge Regression', ridge_best_model),
    ('K-Nearest Neighbors', knn_best_model),
    ('Support Vector Regression', svr_best_model)
]

fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.flatten()

for i, (model_name, model) in enumerate(models):
    y_pred = model.predict(X_test_prep)
    # Transforming back from log scale to original scale
    y_test_exp = np.exp(y_test)
    y_pred_exp = np.exp(y_pred)

    axs[i].scatter(y_test_exp, y_pred_exp, alpha=0.2)
```

```
    axs[i].plot([y_test_exp.min(), y_test_exp.max()], [y_test_exp.min(),␣
↪y_test_exp.max()], 'k--', lw=2)
    axs[i].set_title(model_name)
    axs[i].set_xlabel('Actual Values')
    axs[i].set_ylabel('Predicted Values')

plt.tight_layout()
plt.savefig("true_vs_predict.png")
plt.show()
```

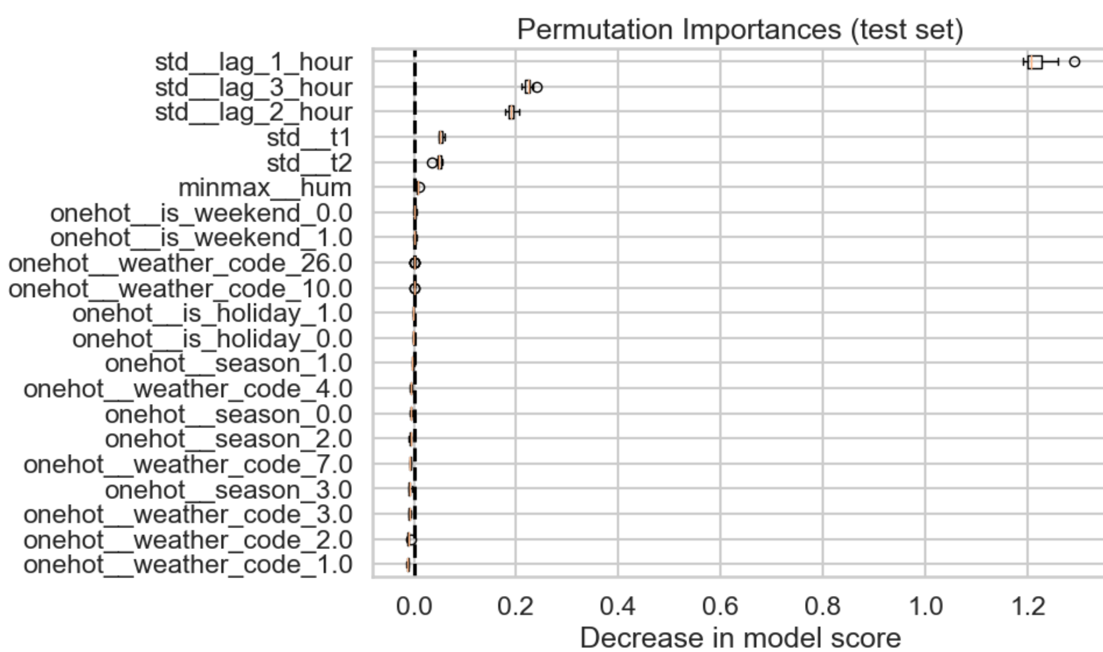


[31]:

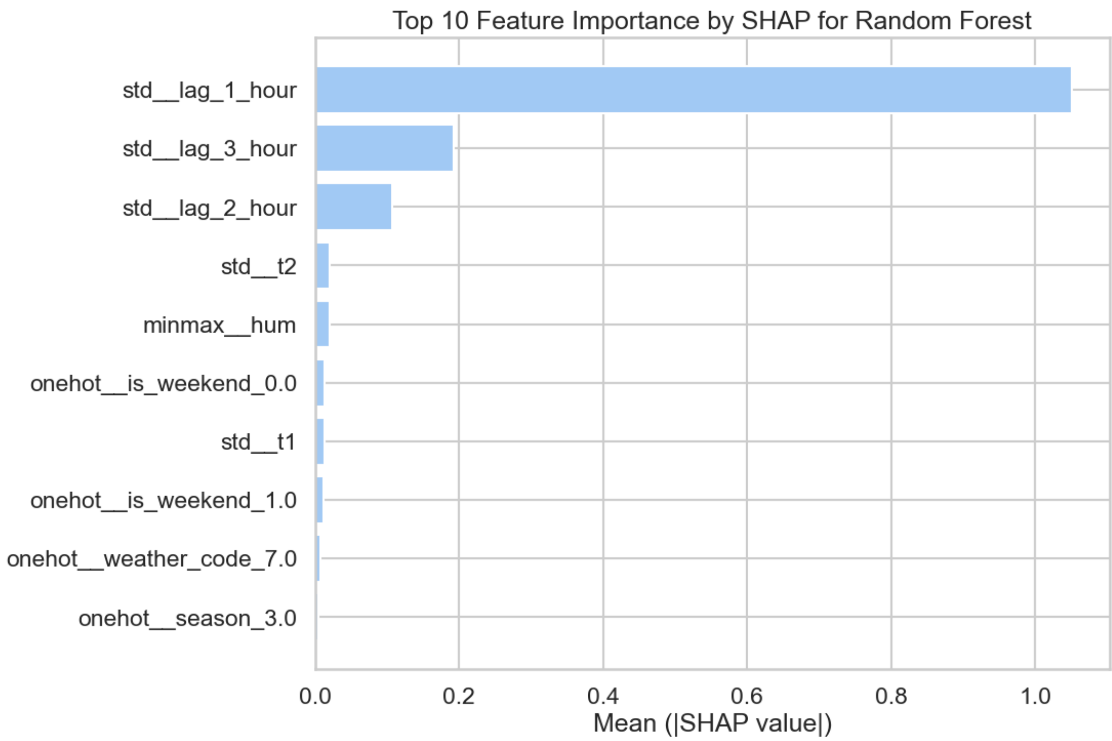
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.inspection import permutation_importance

np.random.seed(88)

test_score = knn_best_model.score(X_test_prep, y_test)
print('Test score =', test_score)
```

```python
# Perform permutation importance
result = permutation_importance(knn_best_model, X_test_prep, y_test,␣
 ↪n_repeats=10, random_state=88, n_jobs=-1)

# Extract feature names from the preprocessor in the pipeline
feature_names = preprocessor.get_feature_names_out()

# Sort the features by importance
sorted_idx = result.importances_mean.argsort()

# Plot
plt.figure(figsize=(10, 6))
plt.boxplot(result.importances[sorted_idx].T, vert=False,␣
 ↪labels=feature_names[sorted_idx])
plt.axvline(0, color='k', linestyle='--')
plt.title("Permutation Importances (test set)")
plt.xlabel('Decrease in model score')
plt.tight_layout()

plt.savefig("permutation_importances.png")
plt.show()
```

```
Test score = 0.8552019393436402
```



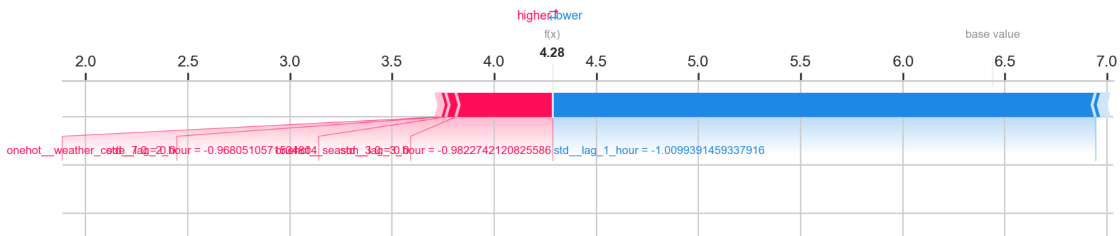
[33]:
```
import shap
import numpy as np
import matplotlib.pyplot as plt

# Initialize the SHAP explainer with your Random Forest model
explainer = shap.TreeExplainer(rf_best_model)

X_sample = shap.utils.sample(X_test_prep, 100)  # Adjust the sample size as␣
↪necessary

# Calculate SHAP values
shap_values = explainer.shap_values(X_sample)

# Calculate the mean absolute SHAP values for each feature
shap_importance = np.abs(shap_values[0]).mean(axis=0) if␣
↪isinstance(shap_values, list) else np.abs(shap_values).mean(axis=0)

# Sorting the feature indices by importance
sorted_indices = np.argsort(shap_importance)

# Plotting
plt.figure(figsize=(10, 8))
plt.barh(np.array(feature_names)[sorted_indices][-10:],␣
↪shap_importance[sorted_indices][-10:])
plt.xlabel('Mean (|SHAP value|)')
plt.title('Top 10 Feature Importance by SHAP for Random Forest')
plt.show()
```



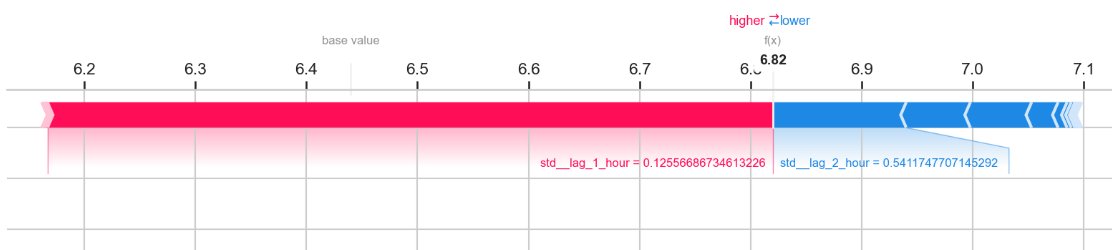
[37]:
```python
# Choose specific data points for the force plot
indices = [0, 100, 200]  # Adjust the indices as per your data
specific_data_points = pd.DataFrame(X_test_prep[indices], columns=feature_names)

# Calculate SHAP values for the specific data points
specific_shap_values = explainer.shap_values(specific_data_points)

# Check if SHAP values are a list (for multi-output models) or an array (for␣
↪single-output models)
is_shap_values_list = isinstance(specific_shap_values, list)

# SHAP force plots for specific data points
for i, index in enumerate(indices):
    shap_value = specific_shap_values[0][i, :] if is_shap_values_list else␣
↪specific_shap_values[i, :]
    expected_value = explainer.expected_value[0] if is_shap_values_list else␣
↪explainer.expected_value

    print(f"SHAP Force Plot for Data Point at Index {index}:")
    shap.force_plot(
        expected_value,
        shap_value,
        specific_data_points.iloc[i],
```

```python
        matplotlib=True,
        link='identity',
        show=True  # This ensures the plot is displayed
    )
```

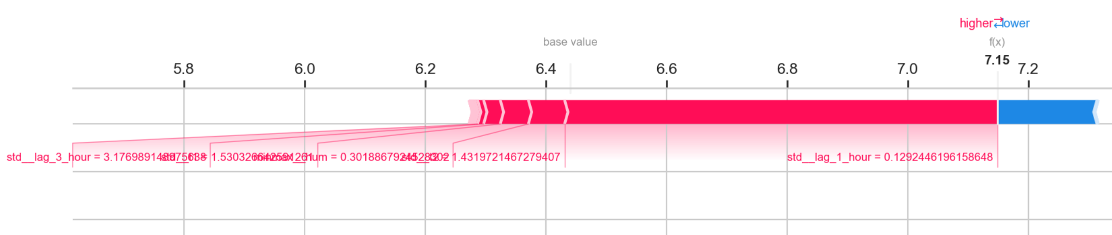
SHAP Force Plot for Data Point at Index 0:



SHAP Force Plot for Data Point at Index 100:



SHAP Force Plot for Data Point at Index 200:



[ ]: