Project Ventoux, part 2: multi feature polynomial regression

A technical overview of part 1 can be found here and a blog post about it can be found here.

Part 2 focuses on taking multiple feature inputs and putting them through a polynomial regression to predict one output. This is heavy on the "finding the right polynomial degree" and less on the "optimizing the parameters," which is something I covered a lot in part 1. Different parts, different skillsets being demonstrated. There is a quick and dirty run of some poly regression based on the output of the MSEs for different polynomial degrees.

The same data table is used in part 1 and part 2. I'm going to carry forward with the same underlying data up through part 4, at which point I'll decide on a new path forward.

Importing data

libraries and methods

In [1]:
```
#I know best practice is to use tian for pandas but I chose the name of a fa
import pandas as tian #named after a DC zoo panda
tian.set_option('display.max_columns', None) #invoking pandas setting I alwa

#I also diverged from numpy as lumpnump because thismakes me laugh and still
import numpy as lumpnump
import matplotlib.pyplot as plt
import os
from sklearn import metrics, linear_model, utils, preprocessing, model_selec
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_scor
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit
from sklearn.preprocessing import StandardScaler, MinMaxScaler, RobustScaler
from scipy.stats import norm, pearsonr
import math
import copy
```

File import

In [2]:
```
top_tracks_stem = os.getenv('TOP_TRACKS_STEM')

toptracks_bpmeta_matrix_csv = f'{top_tracks_stem}/toptracks_bpmeta_matrix'
toptracks_bpmeta_matrix_df_csv = f'{top_tracks_stem}/toptracks_bpmeta_matrix
toptracks_bpmeta_matrix_df = tian.read_csv(toptracks_bpmeta_matrix_df_csv)
toptracks_bpmeta_matrix = lumpnump.genfromtxt(toptracks_bpmeta_matrix_csv, d
```

In [3]:
```
toptracks_bpmeta_matrix_df.head()
```

Out[3]:

	isrc_numeric	release_id	release_date	release_year	release_month	label_id
0	2613012100395	3650447	20211001	2021	10	100342
1	471072337324	4221505	20230818	2023	8	106937
2	72126162358275	4181814	20230707	2023	7	90543
3	72126162358279	4181814	20230707	2023	7	90543
4	72126162358286	4181814	20230707	2023	7	90543

Normalization functions

I'm going to use two different normalization techniques from the Scikit Learn machine learning library.

StandardScaler: one of the most common but also allegedly not great with outliers
PowerTransformer: less common but allegedly better equipped for outliers

The goal is to just see what everything looks like under two different types of normalization

StandardScaler function

Uses StandardScaler() from Scikit Learn to center data around 0, with negative and positive values allowed. StandardScaler is considered sensitive to large outltiers.

This function takes train, test, and validate data as inputs and outputs scaled data.

In [4]:
```
def standard_scaling(x_train, x_validate, x_test, y_train, y_validate, y_tes
    x_features_scaler = StandardScaler()
    y_output_scaler = StandardScaler()

    # Run on training data
    x_train_scaled = x_features_scaler.fit_transform(x_train)
    y_train_scaled = y_output_scaler.fit_transform(y_train)

    # Run on validation data
    x_validate_scaled = x_features_scaler.transform(x_validate)
    y_validate_scaled = y_output_scaler.transform(y_validate)
```

```
    # Run on test data
    x_test_scaled = x_features_scaler.transform(x_test)
    y_test_scaled = y_output_scaler.transform(y_test)

    return x_train_scaled, y_train_scaled, x_validate_scaled, y_validate_sca
```

PowerTransformer function

Applies a power transformation to each feature. This is supposed to make the data a bit more Gaussian esque to stabilize variance and minimize skewness.

"...applies a power transformation to each feature to make the data more Gaussian-like in order to stabilize variance and minimize skewness."--from Scitkit Learn documentation

In [5]:
```
def power_scaling(x_train, x_validate, x_test, y_train, y_validate, y_test):
    x_features_scaler = PowerTransformer()
    y_output_scaler = PowerTransformer()

    # Run on training data
    x_train_scaled = x_features_scaler.fit_transform(x_train)
    y_train_scaled = y_output_scaler.fit_transform(y_train.reshape(-1, 1))

    # Run on validation data
    x_validate_scaled = x_features_scaler.transform(x_validate)
    y_validate_scaled = y_output_scaler.transform(y_validate.reshape(-1, 1))

    # Run on test data
    x_test_scaled = x_features_scaler.transform(x_test)
    y_test_scaled = y_output_scaler.transform(y_test.reshape(-1, 1))

    return x_train_scaled, y_train_scaled, x_validate_scaled, y_validate_sca
```

Preparing data for regression

Creating dictionary and mapping references

Just working with the genres in this public part because it's easier to create a flow to show. I'll work on ways to display some of the more insane data dives I do in private.

In [6]:
```
toptracks_bpmeta_matrix_df_columns = toptracks_bpmeta_matrix_df.columns

toptracks_bpmeta_matrix_df_dict = {name: idx for idx, name in enumerate(topt
toptracks_bpmeta_matrix_df_dict


subset_genres = [6, 90, 93]
subset_genres_dict = {
    6: "Techno (Peak Time/Driving)",
```

```
    90: "Melodic House & Techno",
    93: "Organic House / Downtempo"
}


# grabs the index values for the metrics, y outout, and values for IDs that
core_x_features = [16, 17, 18, 19, 20, 21, 22, 23]
core_y_target = 28
core_id_values = [0, 3, 5, 7]


# these are ratios needed for different parts of the train/test/split flow
train_ratio = 0.6
validate_ratio = 0.5
test_validate_ratio = 0.4
```

Splitting and normalizing

Going to very clearly breakout the data for each normalization type so I can cleanly trace errors. When you're an absolute export, a lot of this can be consolidated but I'm finding it easier to trace and fix errors when they're isolated in their own copies of datasets.

Standard Scaling

In [8]:
```
standard_scaler_matrix = toptracks_bpmeta_matrix
standard_scaler_ids = standard_scaler_matrix[:, core_id_values]
standard_scaler_x_features = standard_scaler_matrix[:, core_x_features]
standard_scaler_y_output = standard_scaler_matrix[:, core_y_target].reshape(

# training and test/validate split
x_train, x_temp, y_train, y_temp, ids_train, ids_temp = train_test_split(sta
                    standard_scaler_y_output, standard_scaler_ids, test_size

#test/validate split
x_validate, x_test, y_validate, y_test, ids_validate, ids_test = train_test_
                                                                 test_size=va



x_train_standard_scaled, y_train_standard_scaled, x_validate_standard_scaled
```

In [9]:
```
#somewhere along the way my y values became tuples and I was over it so I'm
y_train_standard_scaled = lumpnump.array(y_train_standard_scaled).reshape(-1
y_validate_standard_scaled = lumpnump.array(y_validate_standard_scaled).resh
y_test_standard_scaled = lumpnump.array(y_test_standard_scaled).reshape(-1,


#taking all the scaled and not scaled data and putting it back in one matrix
standard_scaling_train = lumpnump.hstack((ids_train, x_train_standard_scaled
standard_scaling_validate = lumpnump.hstack((ids_validate, x_validate_standa
standard_scaling_test = lumpnump.hstack((ids_test, x_test_standard_scaled, y

# Splits size check
print("train:", standard_scaling_train.shape)
```

```
print("validate:", standard_scaling_validate.shape)
print("test:", standard_scaling_test.shape)
```

```
train: (57196, 13)
validate: (19066, 13)
test: (19066, 13)
```

Power Scaling

In [14]:
```
power_scaler_matrix = toptracks_bpmeta_matrix
power_scaler_ids = power_scaler_matrix[:, core_id_values]
power_scaler_x_features = power_scaler_matrix[:, core_x_features]
power_scaler_y_output = power_scaler_matrix[:, core_y_target].reshape(-1, 1)

# training and test/validate split
x_train, x_temp, y_train, y_temp, ids_train, ids_temp = train_test_split(pow
                    power_scaler_y_output, power_scaler_ids, test_size=test_

#test/validate split
x_validate, x_test, y_validate, y_test, ids_validate, ids_test = train_test_
                                                                 test_size=va



x_train_power_scaled, y_train_power_scaled, x_validate_power_scaled, y_valic

#somewhere along the way my y values became tuples and I was over it so I'm
y_train_power_scaled = lumpnump.array(y_train_power_scaled).reshape(-1, 1)
y_validate_power_scaled = lumpnump.array(y_validate_power_scaled).reshape(-1
y_test_power_scaled = lumpnump.array(y_test_power_scaled).reshape(-1, 1)


#taking all the scaled and not scaled data and putting it back in one matrix
power_scaling_train = lumpnump.hstack((ids_train, x_train_power_scaled, y_tr
power_scaling_validate = lumpnump.hstack((ids_validate, x_validate_power_sca
power_scaling_test = lumpnump.hstack((ids_test, x_test_power_scaled, y_test_

# Splits size check
print("train:", power_scaling_train.shape)
print("validate:", power_scaling_validate.shape)
print("test:", power_scaling_test.shape)
```

```
train: (57196, 13)
validate: (19066, 13)
test: (19066, 13)
```

defining polynomial regression and plot functions

This takes train and validate data as inputs and an array of degrees and outputs the mean squared errors and then I smash them together to create some graphs that were inspired by code from deeplearning.ai

In [15]:
```
def poly_degree_regress(x_train, y_train, x_validate, y_validate, degrees):
    train_mean_squared_errors = []
    validate_mean_squared_errors = []
```

```
    for degree in degrees:
        polynomial = PolynomialFeatures(degree, include_bias=False)

        # Train polynomial scaling and regression
        poly_xtrain_mapped = polynomial.fit_transform(x_train)
        scaler = StandardScaler().fit(poly_xtrain_mapped)
        poly_xtrain_mapped_scaled = scaler.transform(poly_xtrain_mapped)

        model = linear_model.LinearRegression()
        model.fit(poly_xtrain_mapped_scaled, y_train)

        # Train MSE
        train_y_hat = model.predict(poly_xtrain_mapped_scaled)
        train_mean_squared_error = mean_squared_error(y_train, train_y_hat)
        train_mean_squared_errors.append(train_mean_squared_error)

        # Validate polynomial scaling and regression
        poly_xvalidate_mapped = polynomial.transform(x_validate)
        poly_xvalidate_mapped_scaled = scaler.transform(poly_xvalidate_mappe

        # Validate MSE
        validate_y_hat = model.predict(poly_xvalidate_mapped_scaled)
        validate_mean_squared_error = mean_squared_error(y_validate, validat
        validate_mean_squared_errors.append(validate_mean_squared_error)

    return train_mean_squared_errors, validate_mean_squared_errors
```

In [16]:
```
#I was trying to sort through weird plotting due to MSE values that were all
#and then side by side. I did this over time and should have found a way to
def plot_train_cv_mses(degrees, mses, title):
    plt.plot(degrees, mses, marker="s", c='cornflowerblue', label='train MSE
    plt.title(title)
    plt.xlabel("Poly degrees")
    plt.ylabel("MSEs")
    plt.legend()
    plt.show()


def train_validate_mses_2plots(degrees, train_mses, validate_mses, title):
    plt.plot(degrees, train_mses, marker="s", c='cornflowerblue', label='tra
    plt.plot(degrees, validate_mses, marker="P", c='fuchsia', label='validat
    plt.title(title)
    plt.xlabel("Poly degrees")
    plt.ylabel("MSEs")
    plt.legend()
    plt.show()
```

Running and graphing different normalization methods

Standard Scaling

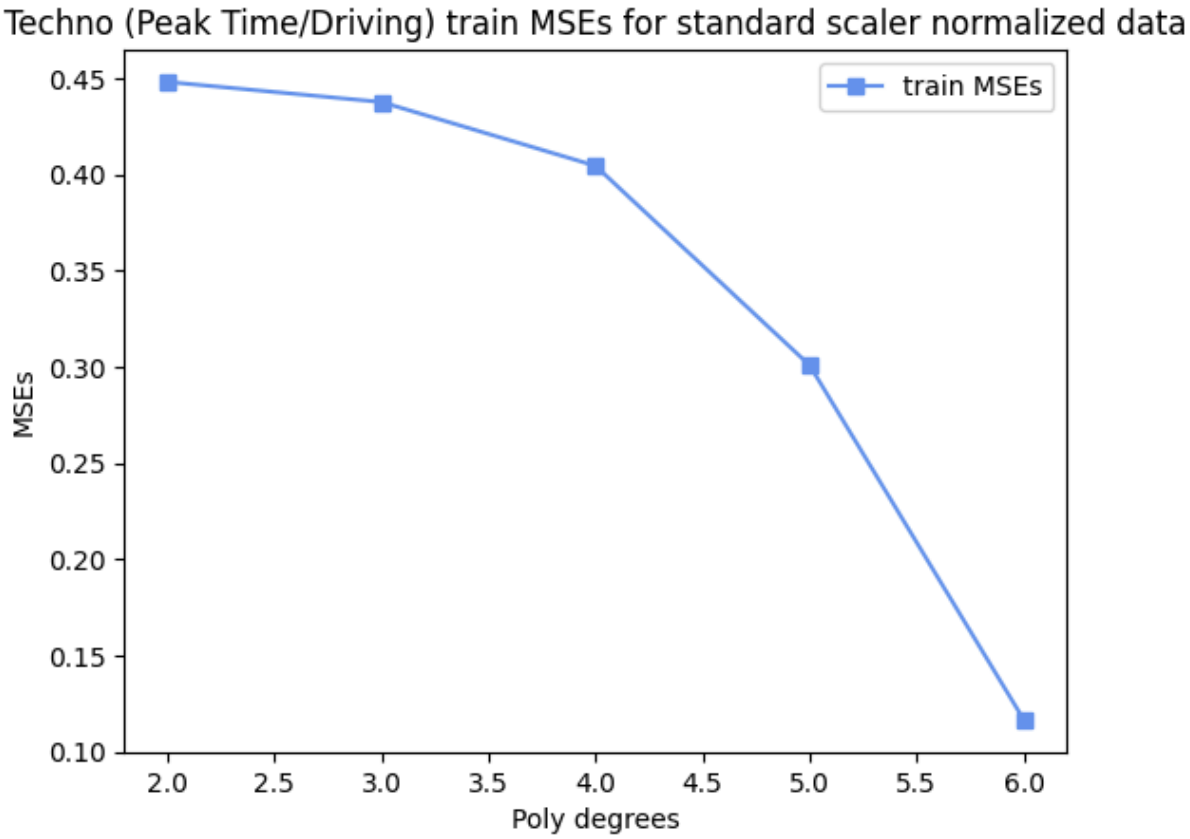
I wrote techno and melodic as very separate things, with the intent to combine into one function to make this cleaner. I kept it this way in the end because it was easier for me to review and I didn't want to spend a lot more time on something that might have minimal value

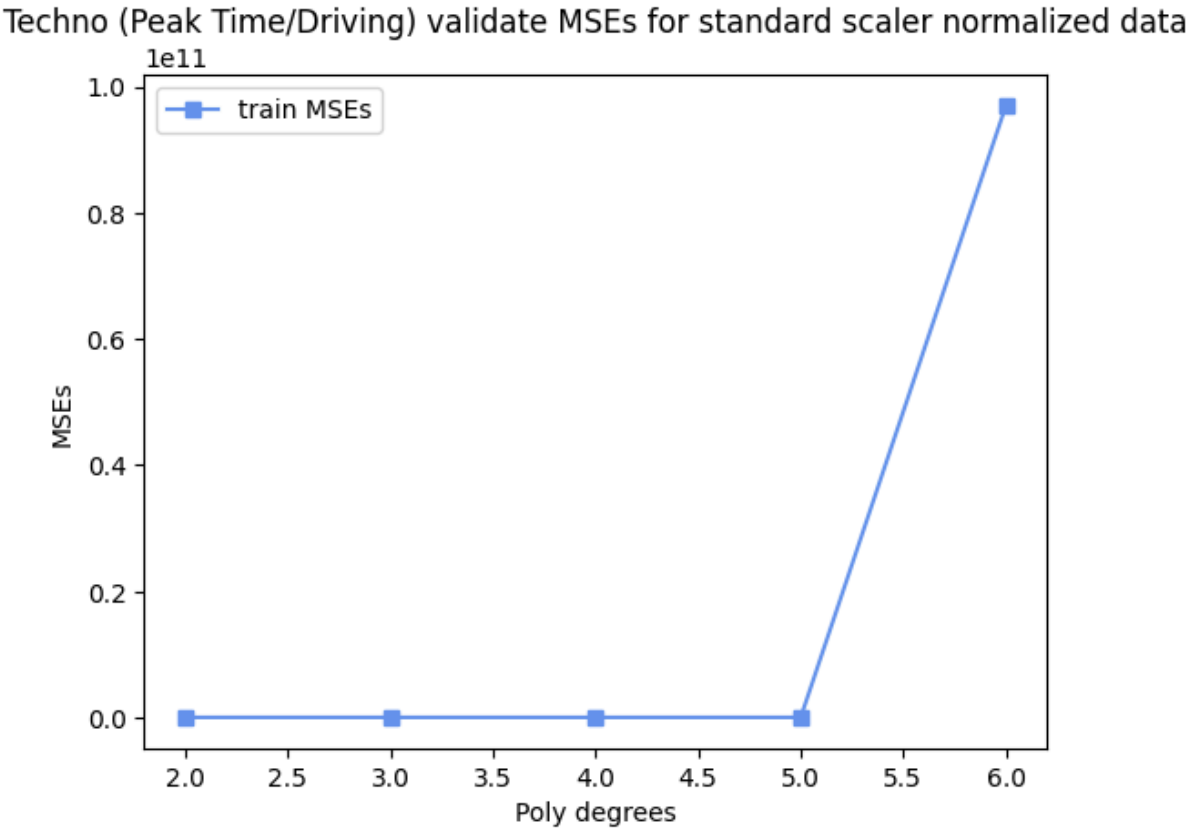
Techno

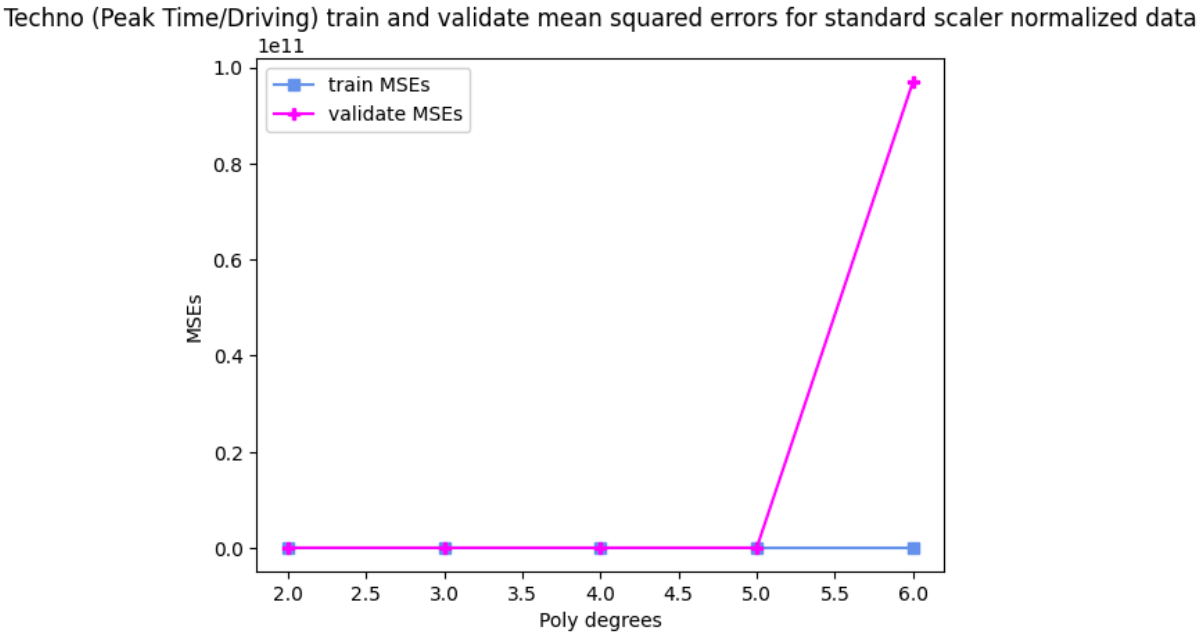
In [17]:
```
techno_degrees = [2, 3, 4, 5, 6]
techno_genre_mask = (standard_scaling_train[:, 3] == 6.)
techno_genre_validate_mask = (standard_scaling_validate[:, 3] == 6.)
techno_genre_test_mask = (standard_scaling_test[:, 3] == 6.)


techno_xtrain = standard_scaling_train[techno_genre_mask, 4:12]
techno_ytrain = standard_scaling_train[techno_genre_mask, 12]

techno_xvalidate = standard_scaling_validate[techno_genre_validate_mask, 4:1
techno_yvalidate = standard_scaling_validate[techno_genre_validate_mask, 12]

techno_xtest = standard_scaling_test[techno_genre_test_mask, 4:12]
techno_ytest = standard_scaling_test[techno_genre_test_mask, 12]

techno_train_mses, techno_validate_mses = poly_degree_regress(techno_xtrain,

# Print results
print("Techno training MSEs:", techno_train_mses)
print("Techno validation MSEs:", techno_validate_mses)
```

```
Techno training MSEs: [0.44801381584391653, 0.4375629750158582, 0.4043895716
0179806, 0.3006896289291029, 0.11637937298236806]
Techno validation MSEs: [0.5254801435286968, 0.5692225595450181, 5.531477722
096978, 187864.0890104906, 96997446950.79507]
```

In [18]:
```
techno = subset_genres_dict[6]
techno_title = f"{techno} train MSEs for standard scaler normalized data"
validate_title =  f"{techno} validate MSEs for standard scaler normalized da
techno_overall_title = f"{techno} train and validate mean squared errors for
plot_train_cv_mses(techno_degrees, techno_train_mses, techno_title)
plot_train_cv_mses(techno_degrees, techno_validate_mses, validate_title)

train_validate_mses_2plots(techno_degrees, techno_train_mses, techno_validat
```


Techno (Peak Time/Driving) train MSEs for standard scaler normalized data
train MSEs
MSEs
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Poly degrees

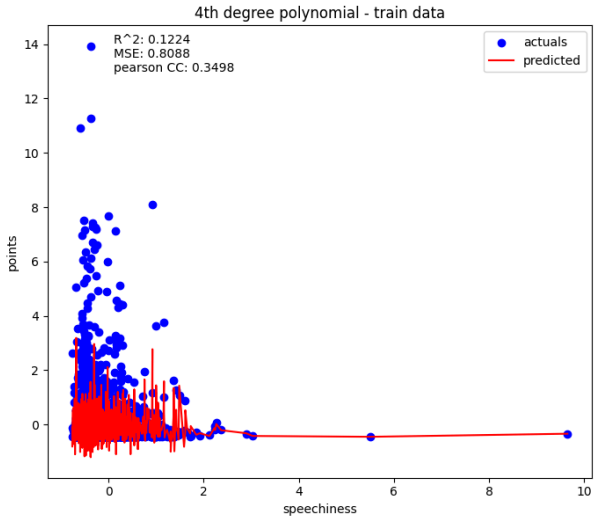

Techno (Peak Time/Driving) validate MSEs for standard scaler normalized data
1e11
train MSEs
MSEs
1.0
0.8
0.6
0.4
0.2
0.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Poly degrees

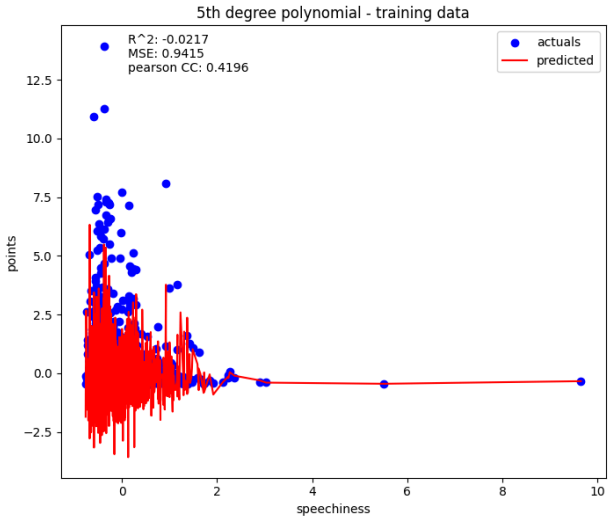


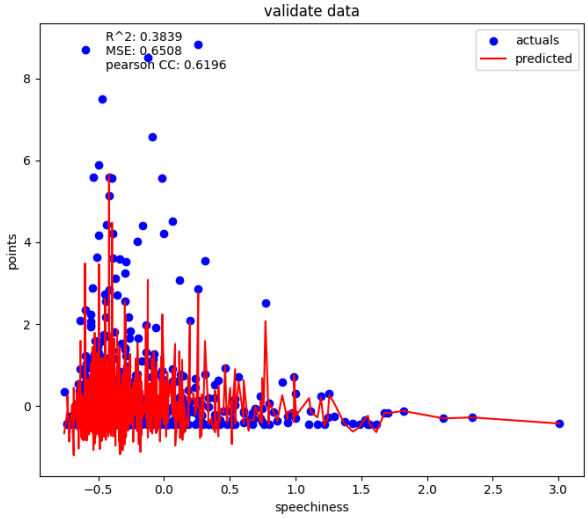
In [19]:
```
techno_poly4 = PolynomialFeatures(degree=4)
techno_x_poly4_train = techno_poly4.fit_transform(techno_xtrain)
techno_x_poly4_validate = techno_poly4.fit_transform(techno_xvalidate)
techno_x_poly4_test = techno_poly4.fit_transform(techno_xtest)

techno_model4 = linear_model.LinearRegression()
techno_model4.fit(techno_x_poly4_train, techno_ytrain)
y_poly_pred4_train = techno_model4.predict(techno_x_poly4_train)

techno_model4.fit(techno_x_poly4_validate, techno_yvalidate)
y_poly_pred4_validate = techno_model4.predict(techno_x_poly4_validate)

techno_model4.fit(techno_x_poly4_test, techno_ytest)
y_poly_pred4_test = techno_model4.predict(techno_x_poly4_test)

techno_sorted_indices_train = lumpnump.argsort(techno_xtrain[:, 3])
techno_sorted_indices_validate = lumpnump.argsort(techno_xvalidate[:, 3])
techno_sorted_indices_test = lumpnump.argsort(techno_xtest[:, 3])

techno_sorted_xtrain_train = techno_xtrain[techno_sorted_indices_train, 3]
techno_sorted_xtrain_validate = techno_xvalidate[techno_sorted_indices_valic
techno_sorted_xtrain_test = techno_xtest[techno_sorted_indices_test, 3]

techno_sorted_ytrain_train = techno_ytrain[techno_sorted_indices_train]
techno_sorted_ytrain_validate = techno_yvalidate[techno_sorted_indices_valic
techno_sorted_ytrain_test = techno_ytest[techno_sorted_indices_test]

techno_sorted_y_poly_pred4_train = y_poly_pred4_train[techno_sorted_indices_
techno_sorted_y_poly_pred4_validate = y_poly_pred4_validate[techno_sorted_ir
techno_sorted_y_poly_pred4_test = y_poly_pred4_test[techno_sorted_indices_te

# Polynomial regression for degree 5
techno_poly5 = PolynomialFeatures(degree=5)
techno_x_poly5_train = techno_poly5.fit_transform(techno_xtrain)
techno_x_poly5_validate = techno_poly5.fit_transform(techno_xvalidate)
techno_x_poly5_test = techno_poly5.fit_transform(techno_xtest)
```

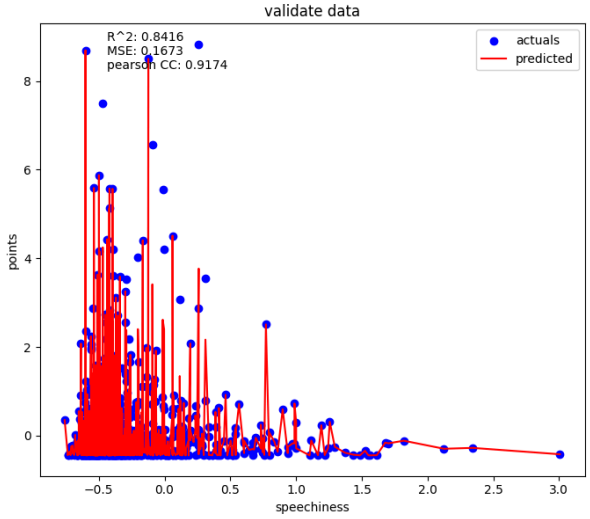
```
# Fitting the polynomial regression model for degree 5
techno_model5 = linear_model.LinearRegression()
techno_model5.fit(techno_x_poly5_train, techno_ytrain)
y_poly_pred5_train = techno_model5.predict(techno_x_poly5_train)

techno_model5.fit(techno_x_poly5_validate, techno_yvalidate)
y_poly_pred5_validate = techno_model5.predict(techno_x_poly5_validate)

techno_model5.fit(techno_x_poly5_test, techno_ytest)
y_poly_pred5_test = techno_model5.predict(techno_x_poly5_test)

techno_sorted_y_poly_pred5_train = y_poly_pred5_train[techno_sorted_indices_
techno_sorted_y_poly_pred5_validate = y_poly_pred5_validate[techno_sorted_ir
techno_sorted_y_poly_pred5_test = y_poly_pred5_test[techno_sorted_indices_te

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(techno_sorted_xtrain_train, techno_sorted_ytrain_train, color='b
plt.plot(techno_sorted_xtrain_train, techno_sorted_y_poly_pred4_train, color
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_techno_train = r2_score(techno_ytrain, y_poly_pred4_train)
mse_4_techno_train = mean_squared_error(techno_ytrain, y_poly_pred4_train)
pearson_corr_4_techno_train, _ = pearsonr(techno_ytrain, y_poly_pred4_train)
plt.figtext(0.1, 0.95, f"R^2: {r2_4_techno_train:.4f}\nMSE: {mse_4_techno_tr

plt.subplot(3, 2, 3)
plt.scatter(techno_sorted_xtrain_validate, techno_sorted_ytrain_validate, co
plt.plot(techno_sorted_xtrain_validate, techno_sorted_y_poly_pred4_validate,
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_techno_validate = r2_score(techno_yvalidate, y_poly_pred4_validate)
mse_4_techno_validate = mean_squared_error(techno_yvalidate, y_poly_pred4_va
pearson_corr_4_techno_validate, _ = pearsonr(techno_yvalidate, y_poly_pred4_
plt.figtext(0.1, 0.62, f"R^2: {r2_4_techno_validate:.4f}\nMSE: {mse_4_techno

plt.subplot(3, 2, 5)
plt.scatter(techno_sorted_xtrain_test, techno_sorted_ytrain_test, color='blu
plt.plot(techno_sorted_xtrain_test, techno_sorted_y_poly_pred4_test, color='
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_techno_test = r2_score(techno_ytest, y_poly_pred4_test)
mse_4_techno_test = mean_squared_error(techno_ytest, y_poly_pred4_test)
pearson_corr_4_techno_test, _ = pearsonr(techno_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_techno_test:.4f}\nMSE: {mse_4_techno_tes
```

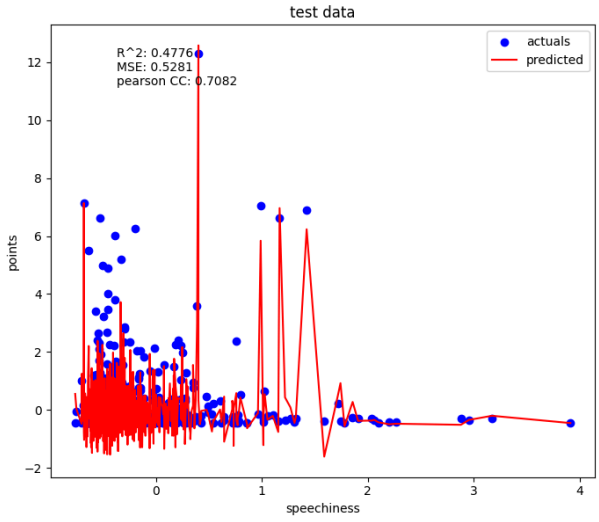
```
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(techno_sorted_xtrain_train, techno_sorted_ytrain_train, color='b
plt.plot(techno_sorted_xtrain_train, techno_sorted_y_poly_pred5_train, color
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_techno_train = r2_score(techno_ytrain, y_poly_pred5_train)
mse_5_techno_train = mean_squared_error(techno_ytrain, y_poly_pred5_train)
pearson_corr_5_techno_train, _ = pearsonr(techno_ytrain, y_poly_pred5_train)
plt.figtext(0.6, 0.95, f"R^2: {r2_5_techno_train:.4f}\nMSE: {mse_5_techno_tr

plt.subplot(3, 2, 4)
plt.scatter(techno_sorted_xtrain_validate, techno_sorted_ytrain_validate, cc
plt.plot(techno_sorted_xtrain_validate, techno_sorted_y_poly_pred5_validate,
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_techno_validate = r2_score(techno_yvalidate, y_poly_pred5_validate)
mse_5_techno_validate = mean_squared_error(techno_yvalidate, y_poly_pred5_va
pearson_corr_5_techno_validate, _ = pearsonr(techno_yvalidate, y_poly_pred5_
plt.figtext(0.6, 0.62, f"R^2: {r2_5_techno_validate:.4f}\nMSE: {mse_5_techno

plt.subplot(3, 2, 6)
plt.scatter(techno_sorted_xtrain_test, techno_sorted_ytrain_test, color='blu
plt.plot(techno_sorted_xtrain_test, techno_sorted_y_poly_pred5_test, color='
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_techno_test = r2_score(techno_ytest, y_poly_pred5_test)
mse_5_techno_test = mean_squared_error(techno_ytest, y_poly_pred5_test)
pearson_corr_5_techno_test, _ = pearsonr(techno_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_techno_test:.4f}\nMSE: {mse_5_techno_tes

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

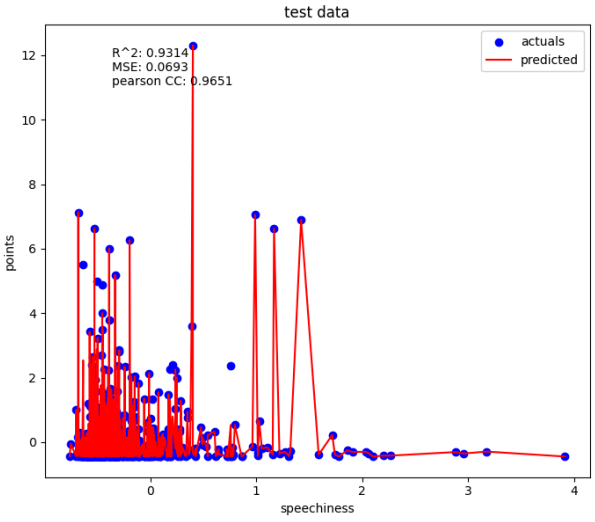












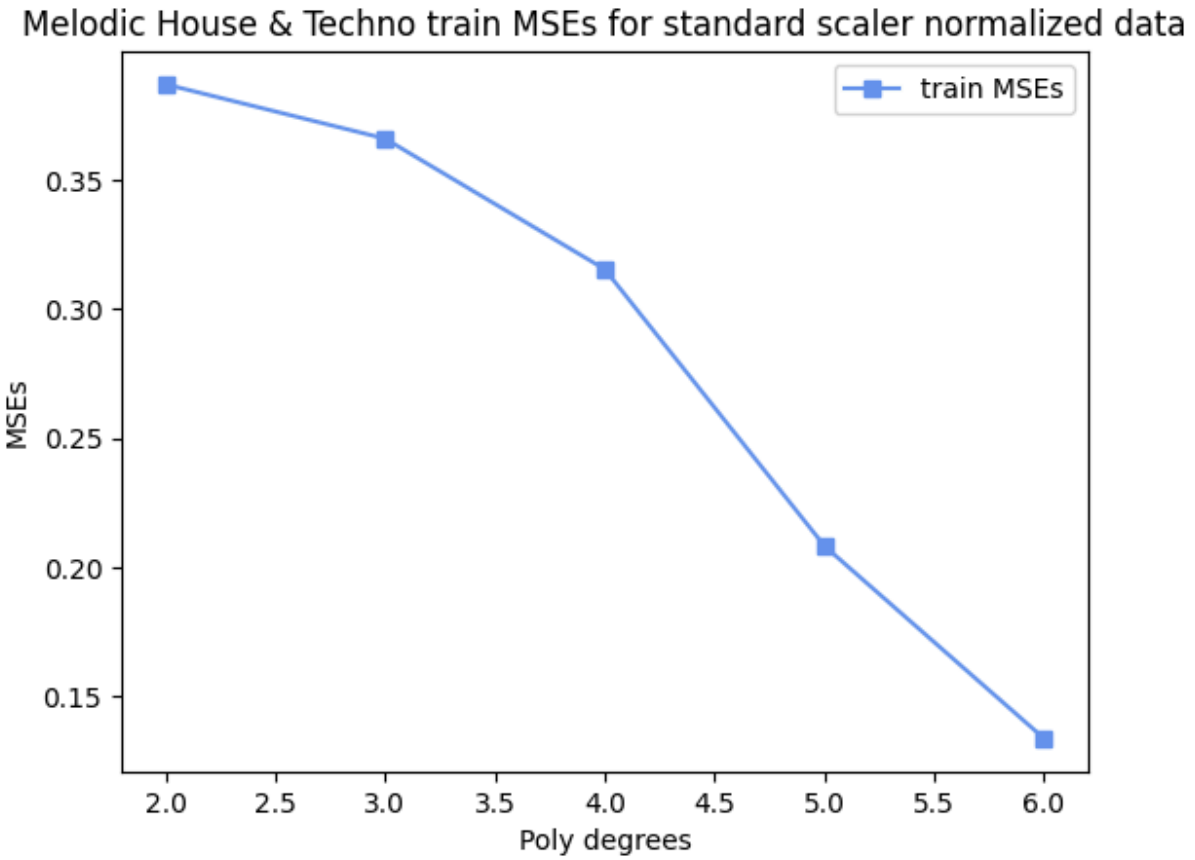
Melodic house and techno

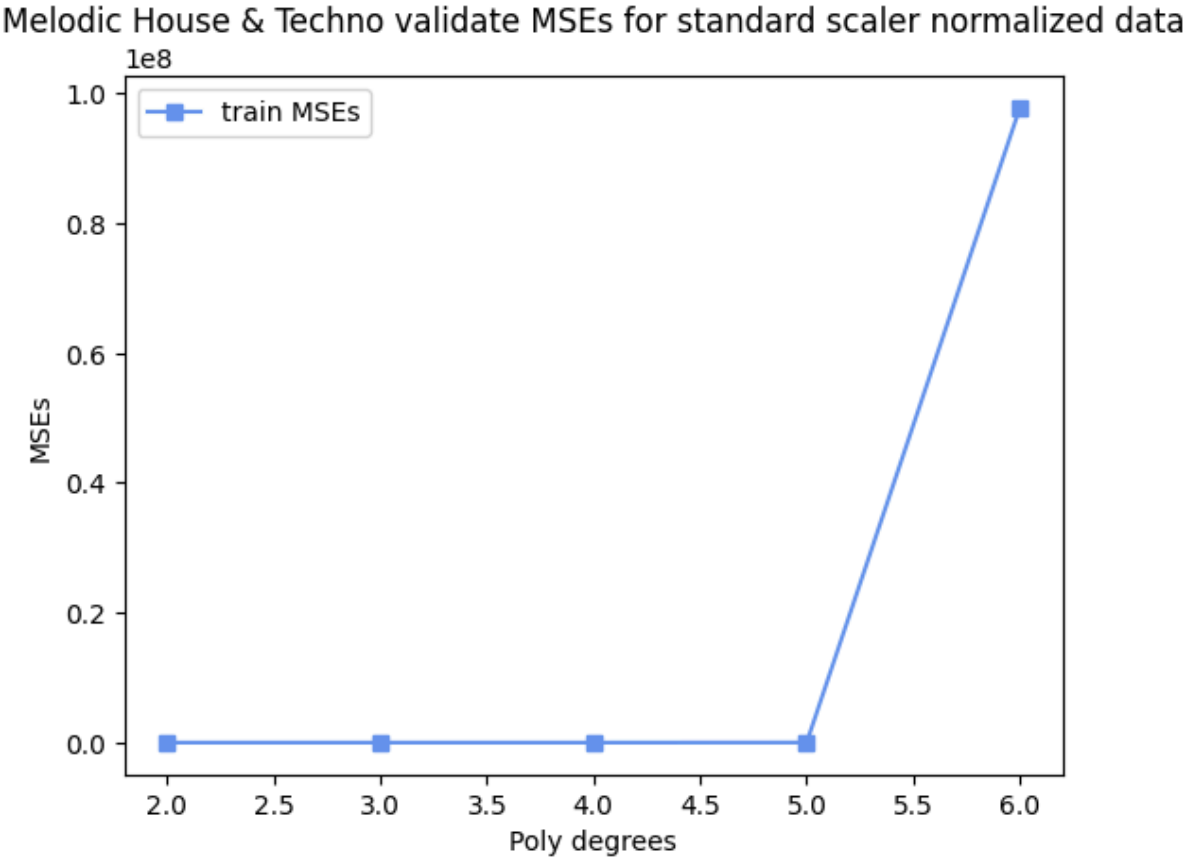
In [20]:
```
melodic_degrees = [2, 3, 4, 5, 6]
melodic_genre_mask = (standard_scaling_train[:, 3] == 90.)
melodic_genre_validate_mask = (standard_scaling_validate[:, 3] == 90.)
melodic_genre_test_mask = (standard_scaling_test[:, 3] == 90.)

melodic_xtrain = standard_scaling_train[melodic_genre_mask, 4:12]
```

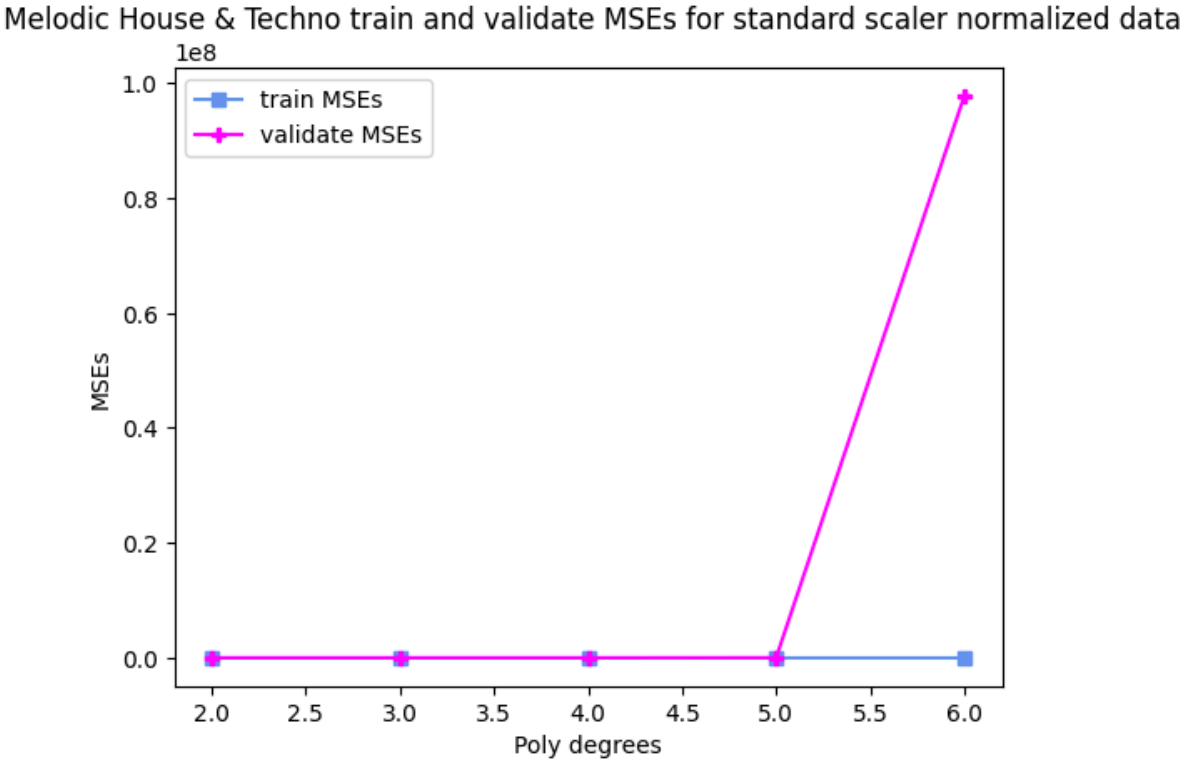
```
melodic_ytrain = standard_scaling_train[melodic_genre_mask, 12]

melodic_xvalidate = standard_scaling_validate[melodic_genre_validate_mask, 4
melodic_yvalidate = standard_scaling_validate[melodic_genre_validate_mask, 1

melodic_xtest = standard_scaling_test[melodic_genre_test_mask, 4:12]
melodic_ytest = standard_scaling_test[melodic_genre_test_mask, 12]

melodic_train_mses, melodic_validate_mses = poly_degree_regress(melodic_xtra

print("melodic training MSEs:", melodic_train_mses)
print("melodic validation MSEs:", melodic_validate_mses)
```

```
melodic training MSEs: [0.38707979987712765, 0.36601986124660324, 0.31541747
14192502, 0.20822987718381777, 0.13365736629943123]
melodic validation MSEs: [0.4728002405862045, 0.4994402316245204, 25.0233593
4889398, 14033.841063327813, 97793264.93353151]
```

In [21]:
```
melodic = subset_genres_dict[90]
melodic_title = f"{melodic} train MSEs for standard scaler normalized data"
melodic_validate_title =  f"{melodic} validate MSEs for standard scaler norm
melodic_overall_title = f" {melodic} train and validate MSEs for standard sc
plot_train_cv_mses(melodic_degrees, melodic_train_mses, melodic_title)
plot_train_cv_mses(melodic_degrees, melodic_validate_mses, melodic_validate_
train_validate_mses_2plots(melodic_degrees, melodic_train_mses, melodic_vali
```





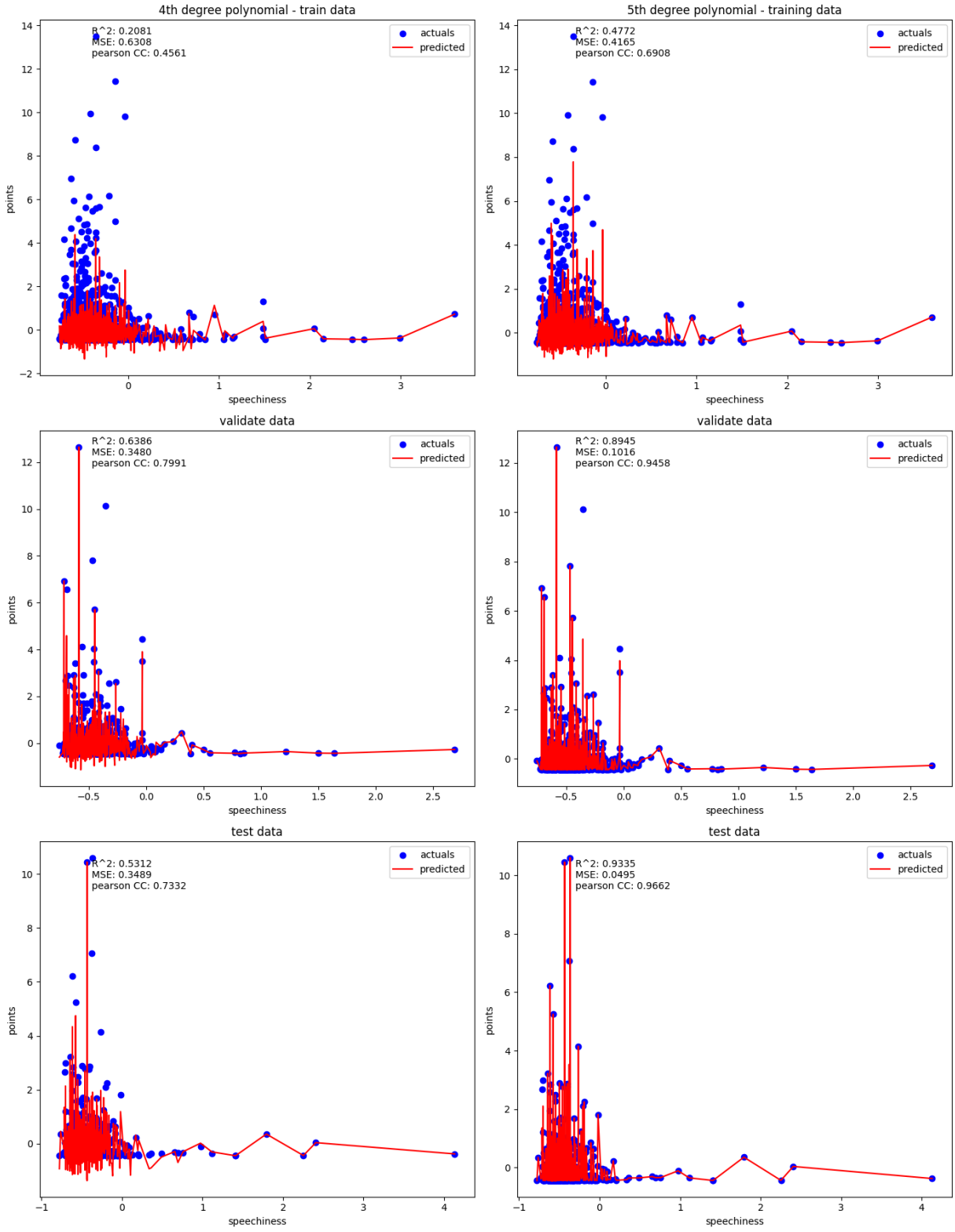


In [22]:
```
melodic_poly4 = PolynomialFeatures(degree=4)
melodic_x_poly4_train = melodic_poly4.fit_transform(melodic_xtrain)
melodic_x_poly4_validate = melodic_poly4.fit_transform(melodic_xvalidate)
melodic_x_poly4_test = melodic_poly4.fit_transform(melodic_xtest)
```

```
melodic_model4 = linear_model.LinearRegression()
melodic_model4.fit(melodic_x_poly4_train, melodic_ytrain)
y_poly_pred4_train = melodic_model4.predict(melodic_x_poly4_train)

melodic_model4.fit(melodic_x_poly4_validate, melodic_yvalidate)
y_poly_pred4_validate = melodic_model4.predict(melodic_x_poly4_validate)

melodic_model4.fit(melodic_x_poly4_test, melodic_ytest)
y_poly_pred4_test = melodic_model4.predict(melodic_x_poly4_test)

melodic_sorted_indices_train = lumpnump.argsort(melodic_xtrain[:, 3])
melodic_sorted_indices_validate = lumpnump.argsort(melodic_xvalidate[:, 3])
melodic_sorted_indices_test = lumpnump.argsort(melodic_xtest[:, 3])

melodic_sorted_xtrain_train = melodic_xtrain[melodic_sorted_indices_train, 3
melodic_sorted_xtrain_validate = melodic_xvalidate[melodic_sorted_indices_va
melodic_sorted_xtrain_test = melodic_xtest[melodic_sorted_indices_test, 3]

melodic_sorted_ytrain_train = melodic_ytrain[melodic_sorted_indices_train]
melodic_sorted_ytrain_validate = melodic_yvalidate[melodic_sorted_indices_va
melodic_sorted_ytrain_test = melodic_ytest[melodic_sorted_indices_test]

melodic_sorted_y_poly_pred4_train = y_poly_pred4_train[melodic_sorted_indice
melodic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[melodic_sorted_
melodic_sorted_y_poly_pred4_test = y_poly_pred4_test[melodic_sorted_indices_

# Polynomial regression for degree 5
melodic_poly5 = PolynomialFeatures(degree=5)
melodic_x_poly5_train = melodic_poly5.fit_transform(melodic_xtrain)
melodic_x_poly5_validate = melodic_poly5.fit_transform(melodic_xvalidate)
melodic_x_poly5_test = melodic_poly5.fit_transform(melodic_xtest)

# Fitting the polynomial regression model for degree 5
melodic_model5 = linear_model.LinearRegression()
melodic_model5.fit(melodic_x_poly5_train, melodic_ytrain)
y_poly_pred5_train = melodic_model5.predict(melodic_x_poly5_train)

melodic_model5.fit(melodic_x_poly5_validate, melodic_yvalidate)
y_poly_pred5_validate = melodic_model5.predict(melodic_x_poly5_validate)

melodic_model5.fit(melodic_x_poly5_test, melodic_ytest)
y_poly_pred5_test = melodic_model5.predict(melodic_x_poly5_test)

melodic_sorted_y_poly_pred5_train = y_poly_pred5_train[melodic_sorted_indice
melodic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[melodic_sorted_
melodic_sorted_y_poly_pred5_test = y_poly_pred5_test[melodic_sorted_indices_

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(melodic_sorted_xtrain_train, melodic_sorted_ytrain_train, color=
plt.plot(melodic_sorted_xtrain_train, melodic_sorted_y_poly_pred4_train, col
plt.xlabel('speechiness')
plt.ylabel('points')
```

```
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_melodic_train = r2_score(melodic_ytrain, y_poly_pred4_train)
mse_4_melodic_train = mean_squared_error(melodic_ytrain, y_poly_pred4_train)
pearson_corr_4_melodic_train, _ = pearsonr(melodic_ytrain, y_poly_pred4_trai
plt.figtext(0.1, 0.95, f"R^2: {r2_4_melodic_train:.4f}\nMSE: {mse_4_melodic_

plt.subplot(3, 2, 3)
plt.scatter(melodic_sorted_xtrain_validate, melodic_sorted_ytrain_validate,
plt.plot(melodic_sorted_xtrain_validate, melodic_sorted_y_poly_pred4_validat
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_melodic_validate = r2_score(melodic_yvalidate, y_poly_pred4_validate)
mse_4_melodic_validate = mean_squared_error(melodic_yvalidate, y_poly_pred4_
pearson_corr_4_melodic_validate, _ = pearsonr(melodic_yvalidate, y_poly_prec
plt.figtext(0.1, 0.62, f"R^2: {r2_4_melodic_validate:.4f}\nMSE: {mse_4_meloc

plt.subplot(3, 2, 5)
plt.scatter(melodic_sorted_xtrain_test, melodic_sorted_ytrain_test, color='b
plt.plot(melodic_sorted_xtrain_test, melodic_sorted_y_poly_pred4_test, color
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_melodic_test = r2_score(melodic_ytest, y_poly_pred4_test)
mse_4_melodic_test = mean_squared_error(melodic_ytest, y_poly_pred4_test)
pearson_corr_4_melodic_test, _ = pearsonr(melodic_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_melodic_test:.4f}\nMSE: {mse_4_melodic_t

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(melodic_sorted_xtrain_train, melodic_sorted_ytrain_train, color=
plt.plot(melodic_sorted_xtrain_train, melodic_sorted_y_poly_pred5_train, col
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_melodic_train = r2_score(melodic_ytrain, y_poly_pred5_train)
mse_5_melodic_train = mean_squared_error(melodic_ytrain, y_poly_pred5_train)
pearson_corr_5_melodic_train, _ = pearsonr(melodic_ytrain, y_poly_pred5_trai
plt.figtext(0.6, 0.95, f"R^2: {r2_5_melodic_train:.4f}\nMSE: {mse_5_melodic_

plt.subplot(3, 2, 4)
plt.scatter(melodic_sorted_xtrain_validate, melodic_sorted_ytrain_validate,
plt.plot(melodic_sorted_xtrain_validate, melodic_sorted_y_poly_pred5_validat
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_melodic_validate = r2_score(melodic_yvalidate, y_poly_pred5_validate)
mse_5_melodic_validate = mean_squared_error(melodic_yvalidate, y_poly_pred5_
pearson_corr_5_melodic_validate, _ = pearsonr(melodic_yvalidate, y_poly_prec
plt.figtext(0.6, 0.62, f"R^2: {r2_5_melodic_validate:.4f}\nMSE: {mse_5_meloc
```

```
plt.subplot(3, 2, 6)
plt.scatter(melodic_sorted_xtrain_test, melodic_sorted_ytrain_test, color='b
plt.plot(melodic_sorted_xtrain_test, melodic_sorted_y_poly_pred5_test, color
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_melodic_test = r2_score(melodic_ytest, y_poly_pred5_test)
mse_5_melodic_test = mean_squared_error(melodic_ytest, y_poly_pred5_test)
pearson_corr_5_melodic_test, _ = pearsonr(melodic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_melodic_test:.4f}\nMSE: {mse_5_melodic_t

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



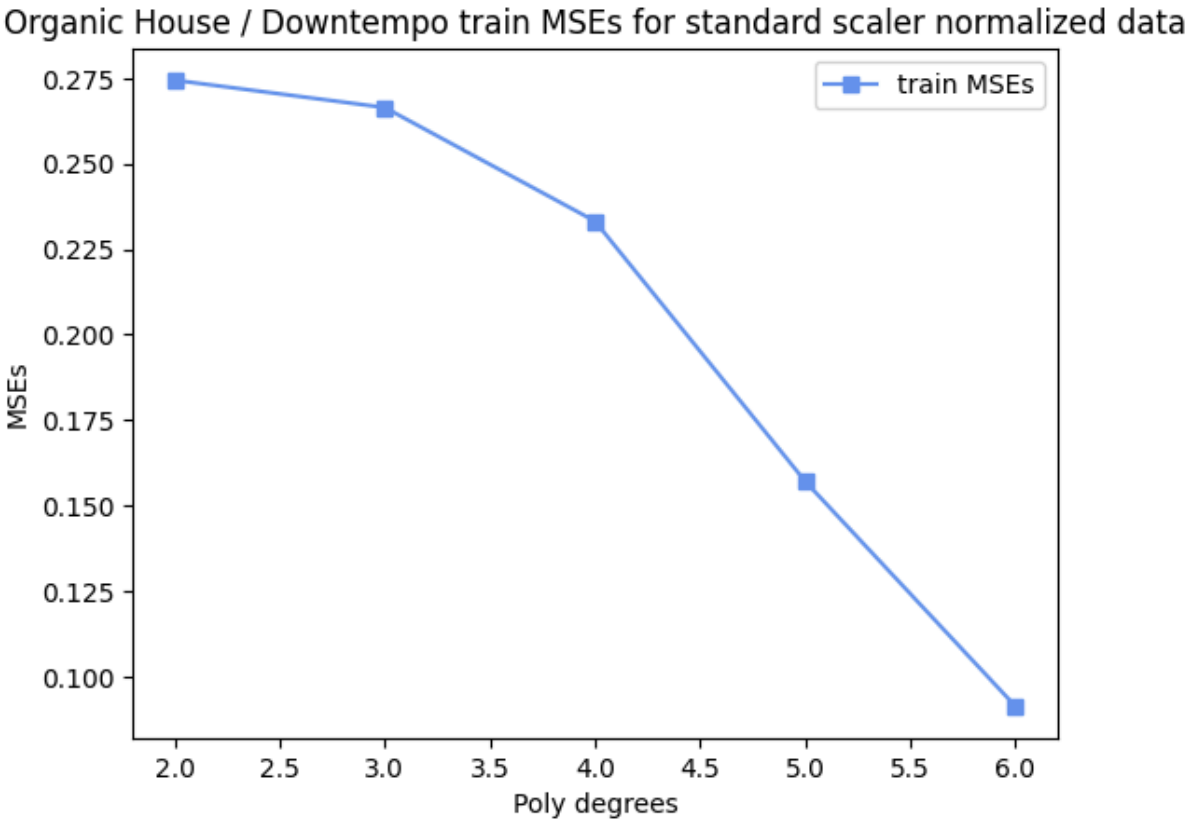
Organic house/down tempo

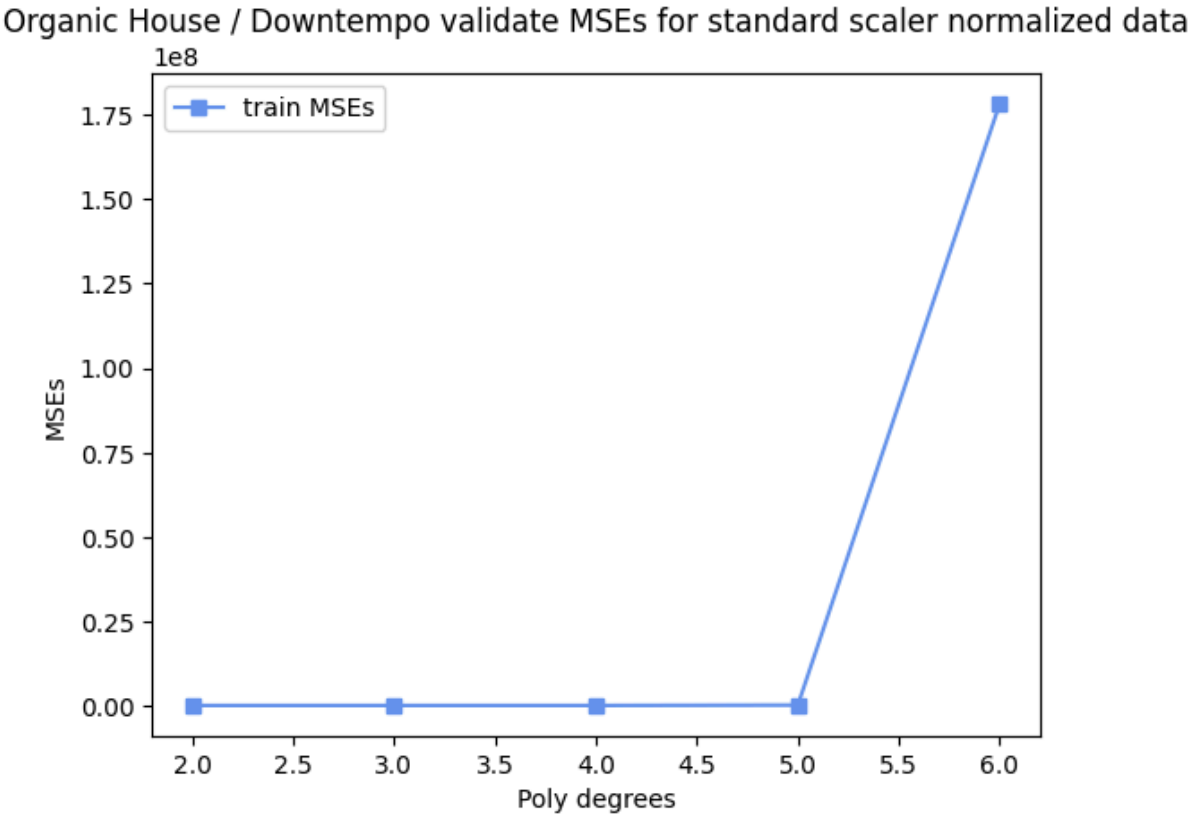
In [23]:
```
organic_degrees = [2, 3, 4, 5, 6]
organic_genre_mask = (standard_scaling_train[:, 3] == 93.)
organic_genre_validate_mask = (standard_scaling_validate[:, 3] == 93.)

organic_xtrain = standard_scaling_train[organic_genre_mask, 4:12]
organic_ytrain = standard_scaling_train[organic_genre_mask, 12]
```

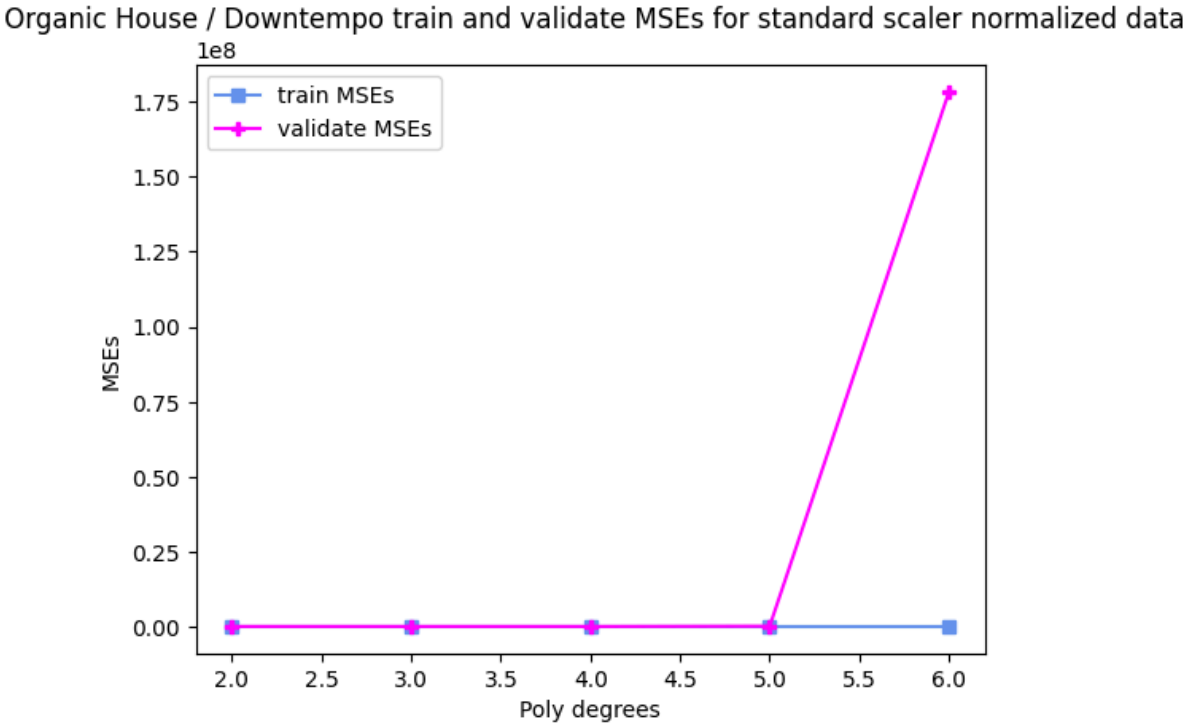
```
organic_xvalidate = standard_scaling_validate[organic_genre_validate_mask, 4
organic_yvalidate = standard_scaling_validate[organic_genre_validate_mask, 1

organic_xtest = standard_scaling_test[organic_genre_validate_mask, 4:12]
organic_ytest = standard_scaling_test[organic_genre_validate_mask, 12]

organic_train_mses, organic_validate_mses = poly_degree_regress(organic_xtra

# Print results
print("organic training MSEs:", organic_train_mses)
print("organic validation MSEs:", organic_validate_mses)
```

```
organic training MSEs: [0.2744470668320491, 0.2664517176839561, 0.2331986949
772892, 0.1570682961457642, 0.09125797394871339]
organic validation MSEs: [0.3736752489156409, 0.39085041097442574, 3.4733239
4293797, 109337.67054748282, 178395347.25449052]
```

In [24]:
```
organic = subset_genres_dict[93]
organic_title = f"{organic} train MSEs for standard scaler normalized data"
organic_validate_title =  f"{organic} validate MSEs for standard scaler norm
organic_overall_title = f" {organic} train and validate MSEs for standard sc
plot_train_cv_mses(organic_degrees, organic_train_mses, organic_title)
plot_train_cv_mses(organic_degrees, organic_validate_mses, organic_validate_
train_validate_mses_2plots(organic_degrees, organic_train_mses, organic_vali
```





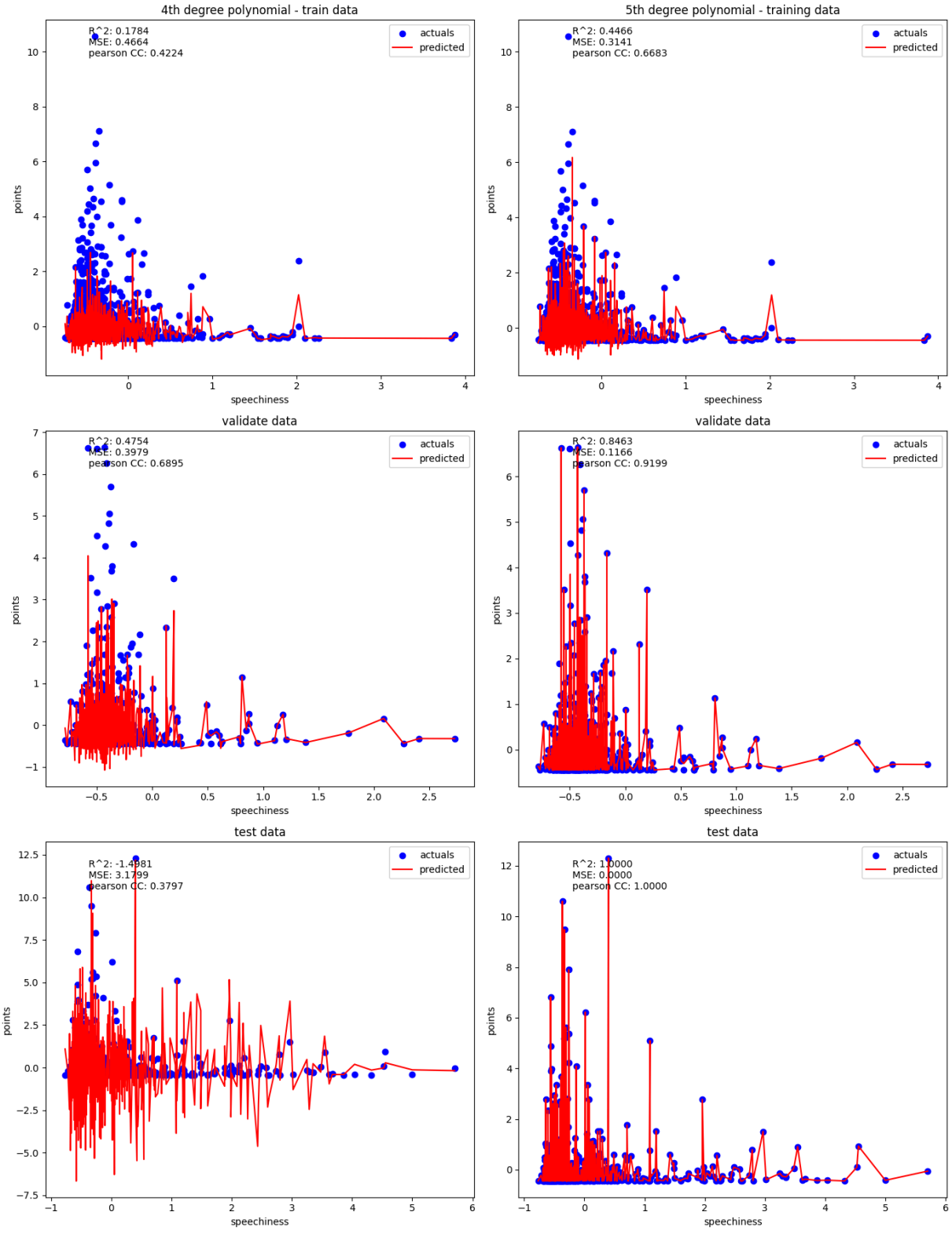


In [25]:
```
organic_poly4 = PolynomialFeatures(degree=4)
organic_x_poly4_train = organic_poly4.fit_transform(organic_xtrain)
organic_x_poly4_validate = organic_poly4.fit_transform(organic_xvalidate)
organic_x_poly4_test = organic_poly4.fit_transform(organic_xtest)

organic_model4 = linear_model.LinearRegression()
organic_model4.fit(organic_x_poly4_train, organic_ytrain)
y_poly_pred4_train = organic_model4.predict(organic_x_poly4_train)
```

```
organic_model4.fit(organic_x_poly4_validate, organic_yvalidate)
y_poly_pred4_validate = organic_model4.predict(organic_x_poly4_validate)

organic_model4.fit(organic_x_poly4_test, organic_ytest)
y_poly_pred4_test = organic_model4.predict(organic_x_poly4_test)

organic_sorted_indices_train = lumpnump.argsort(organic_xtrain[:, 3])
organic_sorted_indices_validate = lumpnump.argsort(organic_xvalidate[:, 3])
organic_sorted_indices_test = lumpnump.argsort(organic_xtest[:, 3])

organic_sorted_xtrain_train = organic_xtrain[organic_sorted_indices_train, 3
organic_sorted_xtrain_validate = organic_xvalidate[organic_sorted_indices_va
organic_sorted_xtrain_test = organic_xtest[organic_sorted_indices_test, 3]

organic_sorted_ytrain_train = organic_ytrain[organic_sorted_indices_train]
organic_sorted_ytrain_validate = organic_yvalidate[organic_sorted_indices_va
organic_sorted_ytrain_test = organic_ytest[organic_sorted_indices_test]

organic_sorted_y_poly_pred4_train = y_poly_pred4_train[organic_sorted_indice
organic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[organic_sorted_
organic_sorted_y_poly_pred4_test = y_poly_pred4_test[organic_sorted_indices_

# Polynomial regression for degree 5
organic_poly5 = PolynomialFeatures(degree=5)
organic_x_poly5_train = organic_poly5.fit_transform(organic_xtrain)
organic_x_poly5_validate = organic_poly5.fit_transform(organic_xvalidate)
organic_x_poly5_test = organic_poly5.fit_transform(organic_xtest)

# Fitting the polynomial regression model for degree 5
organic_model5 = linear_model.LinearRegression()
organic_model5.fit(organic_x_poly5_train, organic_ytrain)
y_poly_pred5_train = organic_model5.predict(organic_x_poly5_train)

organic_model5.fit(organic_x_poly5_validate, organic_yvalidate)
y_poly_pred5_validate = organic_model5.predict(organic_x_poly5_validate)

organic_model5.fit(organic_x_poly5_test, organic_ytest)
y_poly_pred5_test = organic_model5.predict(organic_x_poly5_test)

organic_sorted_y_poly_pred5_train = y_poly_pred5_train[organic_sorted_indice
organic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[organic_sorted_
organic_sorted_y_poly_pred5_test = y_poly_pred5_test[organic_sorted_indices_

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(organic_sorted_xtrain_train, organic_sorted_ytrain_train, color=
plt.plot(organic_sorted_xtrain_train, organic_sorted_y_poly_pred4_train, col
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_organic_train = r2_score(organic_ytrain, y_poly_pred4_train)
```

```
mse_4_organic_train = mean_squared_error(organic_ytrain, y_poly_pred4_train)
pearson_corr_4_organic_train, _ = pearsonr(organic_ytrain, y_poly_pred4_trai
plt.figtext(0.1, 0.95, f"R^2: {r2_4_organic_train:.4f}\nMSE: {mse_4_organic_

plt.subplot(3, 2, 3)
plt.scatter(organic_sorted_xtrain_validate, organic_sorted_ytrain_validate,
plt.plot(organic_sorted_xtrain_validate, organic_sorted_y_poly_pred4_validat
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_organic_validate = r2_score(organic_yvalidate, y_poly_pred4_validate)
mse_4_organic_validate = mean_squared_error(organic_yvalidate, y_poly_pred4_
pearson_corr_4_organic_validate, _ = pearsonr(organic_yvalidate, y_poly_prec
plt.figtext(0.1, 0.62, f"R^2: {r2_4_organic_validate:.4f}\nMSE: {mse_4_orgar

plt.subplot(3, 2, 5)
plt.scatter(organic_sorted_xtrain_test, organic_sorted_ytrain_test, color='b
plt.plot(organic_sorted_xtrain_test, organic_sorted_y_poly_pred4_test, color
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_organic_test = r2_score(organic_ytest, y_poly_pred4_test)
mse_4_organic_test = mean_squared_error(organic_ytest, y_poly_pred4_test)
pearson_corr_4_organic_test, _ = pearsonr(organic_ytest, y_poly_pred4_test)
plt.figtext(0.1, 0.28, f"R^2: {r2_4_organic_test:.4f}\nMSE: {mse_4_organic_t

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(organic_sorted_xtrain_train, organic_sorted_ytrain_train, color=
plt.plot(organic_sorted_xtrain_train, organic_sorted_y_poly_pred5_train, col
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_organic_train = r2_score(organic_ytrain, y_poly_pred5_train)
mse_5_organic_train = mean_squared_error(organic_ytrain, y_poly_pred5_train)
pearson_corr_5_organic_train, _ = pearsonr(organic_ytrain, y_poly_pred5_trai
plt.figtext(0.6, 0.95, f"R^2: {r2_5_organic_train:.4f}\nMSE: {mse_5_organic_

plt.subplot(3, 2, 4)
plt.scatter(organic_sorted_xtrain_validate, organic_sorted_ytrain_validate,
plt.plot(organic_sorted_xtrain_validate, organic_sorted_y_poly_pred5_validat
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_organic_validate = r2_score(organic_yvalidate, y_poly_pred5_validate)
mse_5_organic_validate = mean_squared_error(organic_yvalidate, y_poly_pred5_
pearson_corr_5_organic_validate, _ = pearsonr(organic_yvalidate, y_poly_prec
plt.figtext(0.6, 0.62, f"R^2: {r2_5_organic_validate:.4f}\nMSE: {mse_5_orgar

plt.subplot(3, 2, 6)
plt.scatter(organic_sorted_xtrain_test, organic_sorted_ytrain_test, color='b
plt.plot(organic_sorted_xtrain_test, organic_sorted_y_poly_pred5_test, color
```

```
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_organic_test = r2_score(organic_ytest, y_poly_pred5_test)
mse_5_organic_test = mean_squared_error(organic_ytest, y_poly_pred5_test)
pearson_corr_5_organic_test, _ = pearsonr(organic_ytest, y_poly_pred5_test)
plt.figtext(0.6, 0.28, f"R^2: {r2_5_organic_test:.4f}\nMSE: {mse_5_organic_t

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



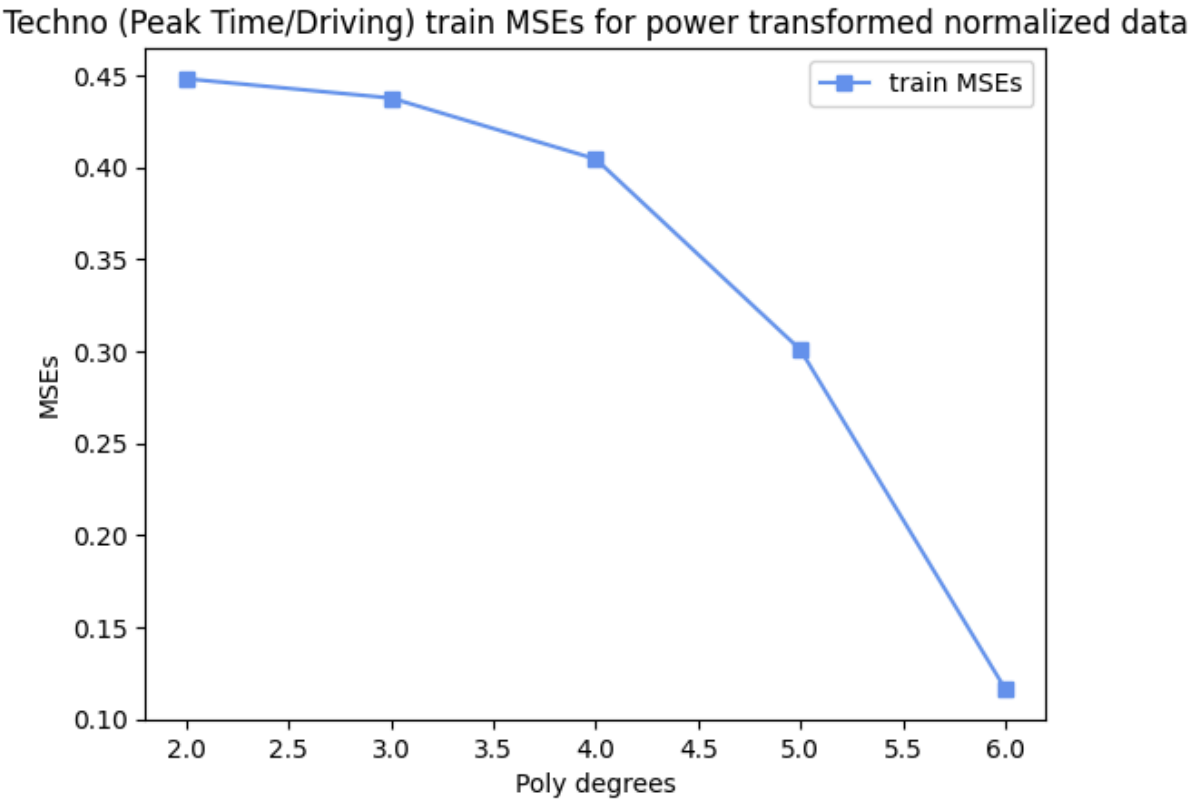
Power Scaling

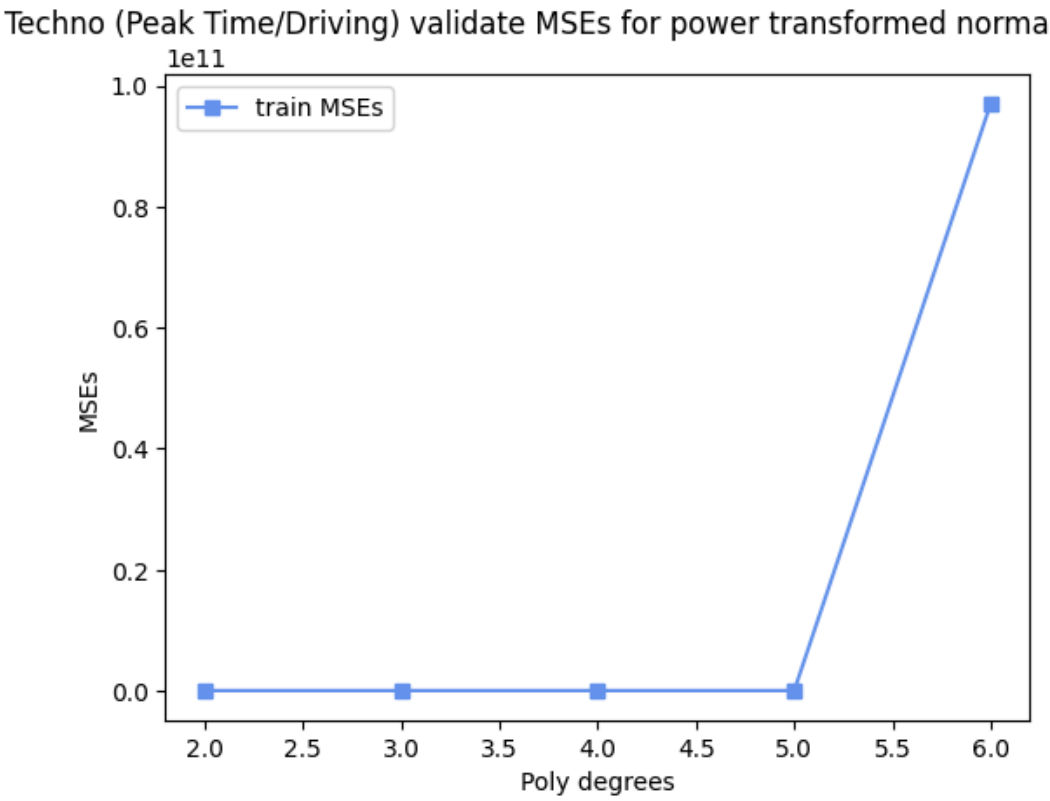
Techno

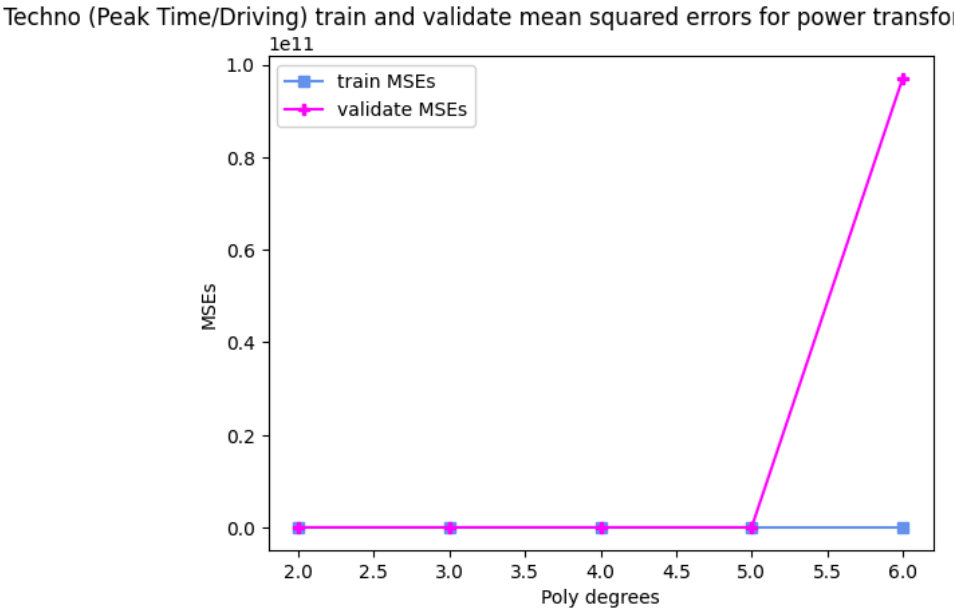
In [26]:
```
power_techno_degrees = [2, 3, 4, 5, 6]
power_techno_genre_mask = (power_scaling_train[:, 3] == 6.)
power_techno_genre_validate_mask = (power_scaling_validate[:, 3] == 6.)
```

```
power_techno_genre_test_mask = (power_scaling_test[:, 3] == 6.)


power_techno_xtrain = power_scaling_train[power_techno_genre_mask, 4:12]
power_techno_ytrain = power_scaling_train[power_techno_genre_mask, 12]

power_techno_xvalidate = power_scaling_validate[power_techno_genre_validate_
power_techno_yvalidate = power_scaling_validate[power_techno_genre_validate_

power_techno_xtest = power_scaling_test[power_techno_genre_test_mask, 4:12]
power_techno_ytest = power_scaling_test[power_techno_genre_test_mask, 12]

power_techno_train_mses, power_techno_validate_mses = poly_degree_regress(pc

# Print results
print("Techno training MSEs:", power_techno_train_mses)
print("Techno validation MSEs:", power_techno_validate_mses)


techno = subset_genres_dict[6]
techno_title = f"{techno} train MSEs for power transformed normalized data"
validate_title =  f"{techno} validate MSEs for power transformed normalized
techno_overall_title = f"{techno} train and validate mean squared errors for
plot_train_cv_mses(techno_degrees, techno_train_mses, techno_title)
plot_train_cv_mses(techno_degrees, techno_validate_mses, validate_title)

train_validate_mses_2plots(techno_degrees, techno_train_mses, techno_validat
```

```
Techno training MSEs: [0.4927375328876941, 0.47117030330210646, 0.4131166882
001141, 0.2776718050641688, 0.061247500670738204]
Techno validation MSEs: [0.47579164415755204, 0.4984695325814922, 0.67761138
99118335, 51.845867628143935, 119932.51069159251]
```





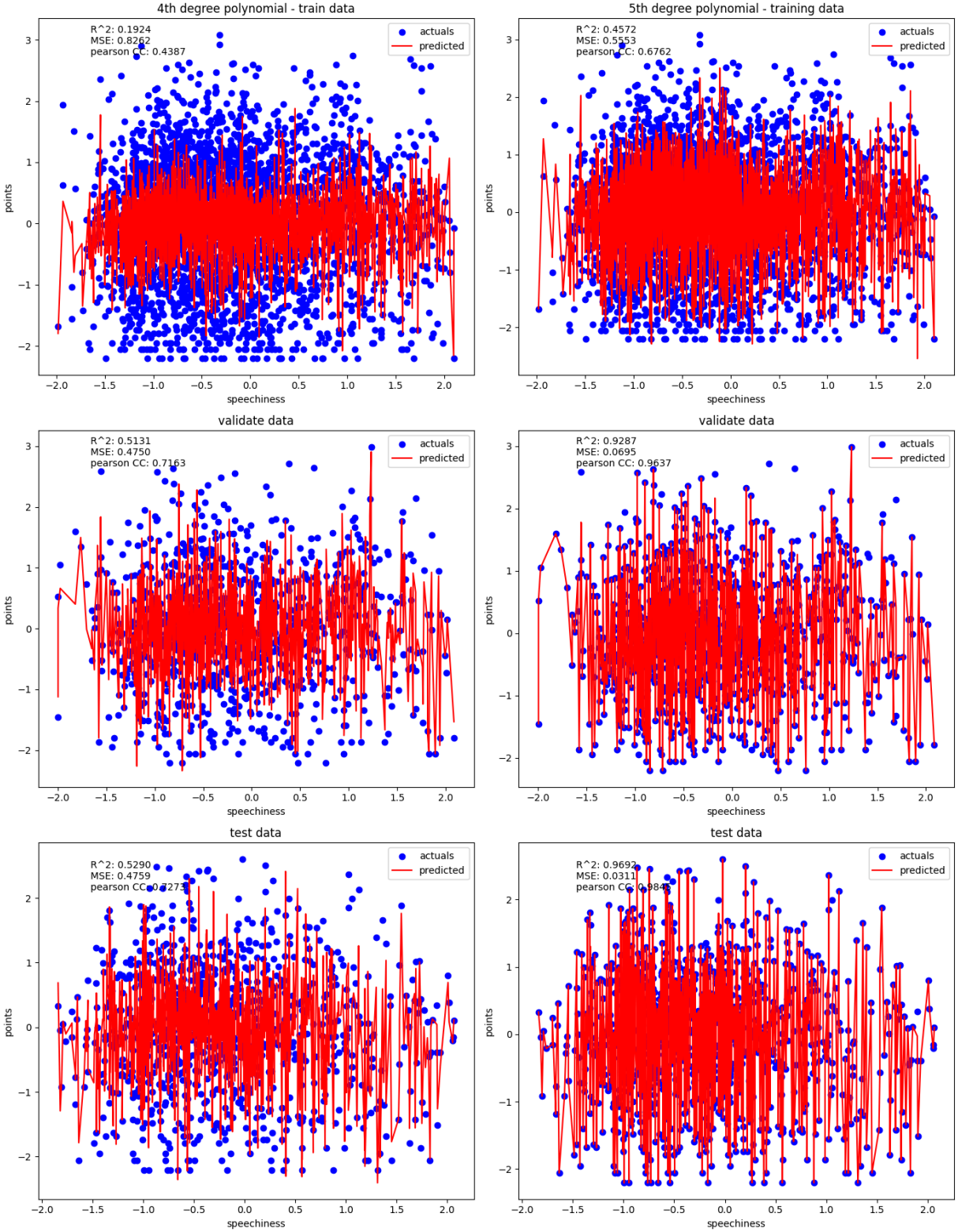


In [27]:
```
power_techno_poly4 = PolynomialFeatures(degree=4)
power_techno_x_poly4_train = power_techno_poly4.fit_transform(power_techno_x
power_techno_x_poly4_validate = power_techno_poly4.fit_transform(power_techr
power_techno_x_poly4_test = power_techno_poly4.fit_transform(power_techno_xt

power_techno_model4 = linear_model.LinearRegression()
power_techno_model4.fit(power_techno_x_poly4_train, power_techno_ytrain)
y_poly_pred4_train = power_techno_model4.predict(power_techno_x_poly4_train)

power_techno_model4.fit(power_techno_x_poly4_validate, power_techno_yvalidat
y_poly_pred4_validate = power_techno_model4.predict(power_techno_x_poly4_val
```

```
power_techno_model4.fit(power_techno_x_poly4_test, power_techno_ytest)
y_poly_pred4_test = power_techno_model4.predict(power_techno_x_poly4_test)

power_techno_sorted_indices_train = lumpnump.argsort(power_techno_xtrain[:,
power_techno_sorted_indices_validate = lumpnump.argsort(power_techno_xvalida
power_techno_sorted_indices_test = lumpnump.argsort(power_techno_xtest[:, 3]

power_techno_sorted_xtrain_train = power_techno_xtrain[power_techno_sorted_i
power_techno_sorted_xtrain_validate = power_techno_xvalidate[power_techno_sc
power_techno_sorted_xtrain_test = power_techno_xtest[power_techno_sorted_inc

power_techno_sorted_ytrain_train = power_techno_ytrain[power_techno_sorted_i
power_techno_sorted_ytrain_validate = power_techno_yvalidate[power_techno_sc
power_techno_sorted_ytrain_test = power_techno_ytest[power_techno_sorted_inc

power_techno_sorted_y_poly_pred4_train = y_poly_pred4_train[power_techno_sor
power_techno_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_tech
power_techno_sorted_y_poly_pred4_test = y_poly_pred4_test[power_techno_sorte

# Polynomial regression for degree 5
power_techno_poly5 = PolynomialFeatures(degree=5)
power_techno_x_poly5_train = power_techno_poly5.fit_transform(power_techno_x
power_techno_x_poly5_validate = power_techno_poly5.fit_transform(power_techr
power_techno_x_poly5_test = power_techno_poly5.fit_transform(power_techno_xt

# Fitting the polynomial regression model for degree 5
power_techno_model5 = linear_model.LinearRegression()
power_techno_model5.fit(power_techno_x_poly5_train, power_techno_ytrain)
y_poly_pred5_train = power_techno_model5.predict(power_techno_x_poly5_train)

power_techno_model5.fit(power_techno_x_poly5_validate, power_techno_yvalidat
y_poly_pred5_validate = power_techno_model5.predict(power_techno_x_poly5_val

power_techno_model5.fit(power_techno_x_poly5_test, power_techno_ytest)
y_poly_pred5_test = power_techno_model5.predict(power_techno_x_poly5_test)

power_techno_sorted_y_poly_pred5_train = y_poly_pred5_train[power_techno_sor
power_techno_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_tech
power_techno_sorted_y_poly_pred5_test = y_poly_pred5_test[power_techno_sorte

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_techno_sorted_xtrain_train, power_techno_sorted_ytrain_tra
plt.plot(power_techno_sorted_xtrain_train, power_techno_sorted_y_poly_pred4_
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_techno_train = r2_score(power_techno_ytrain, y_poly_pred4_train)
mse_4_techno_train = mean_squared_error(power_techno_ytrain, y_poly_pred4_tr
pearson_corr_4_techno_train, _ = pearsonr(power_techno_ytrain, y_poly_pred4_
plt.figtext(0.1, 0.95, f"R^2: {r2_4_techno_train:.4f}\nMSE: {mse_4_techno_tr
```

```
plt.subplot(3, 2, 3)
plt.scatter(power_techno_sorted_xtrain_validate, power_techno_sorted_ytrain_
plt.plot(power_techno_sorted_xtrain_validate, power_techno_sorted_y_poly_pre
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_techno_validate = r2_score(power_techno_yvalidate, y_poly_pred4_validat
mse_4_techno_validate = mean_squared_error(power_techno_yvalidate, y_poly_pr
pearson_corr_4_techno_validate, _ = pearsonr(power_techno_yvalidate, y_poly_
plt.figtext(0.1, 0.62, f"R^2: {r2_4_techno_validate:.4f}\nMSE: {mse_4_techno

plt.subplot(3, 2, 5)
plt.scatter(power_techno_sorted_xtrain_test, power_techno_sorted_ytrain_test
plt.plot(power_techno_sorted_xtrain_test, power_techno_sorted_y_poly_pred4_t
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_techno_test = r2_score(power_techno_ytest, y_poly_pred4_test)
mse_4_techno_test = mean_squared_error(power_techno_ytest, y_poly_pred4_test
pearson_corr_4_techno_test, _ = pearsonr(power_techno_ytest, y_poly_pred4_te
plt.figtext(0.1, 0.28, f"R^2: {r2_4_techno_test:.4f}\nMSE: {mse_4_techno_tes

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_techno_sorted_xtrain_train, power_techno_sorted_ytrain_tra
plt.plot(power_techno_sorted_xtrain_train, power_techno_sorted_y_poly_pred5_
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_techno_train = r2_score(power_techno_ytrain, y_poly_pred5_train)
mse_5_techno_train = mean_squared_error(power_techno_ytrain, y_poly_pred5_tr
pearson_corr_5_techno_train, _ = pearsonr(power_techno_ytrain, y_poly_pred5_
plt.figtext(0.6, 0.95, f"R^2: {r2_5_techno_train:.4f}\nMSE: {mse_5_techno_tr

plt.subplot(3, 2, 4)
plt.scatter(power_techno_sorted_xtrain_validate, power_techno_sorted_ytrain_
plt.plot(power_techno_sorted_xtrain_validate, power_techno_sorted_y_poly_pre
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_techno_validate = r2_score(power_techno_yvalidate, y_poly_pred5_validat
mse_5_techno_validate = mean_squared_error(power_techno_yvalidate, y_poly_pr
pearson_corr_5_techno_validate, _ = pearsonr(power_techno_yvalidate, y_poly_
plt.figtext(0.6, 0.62, f"R^2: {r2_5_techno_validate:.4f}\nMSE: {mse_5_techno

plt.subplot(3, 2, 6)
plt.scatter(power_techno_sorted_xtrain_test, power_techno_sorted_ytrain_test
plt.plot(power_techno_sorted_xtrain_test, power_techno_sorted_y_poly_pred5_t
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
```

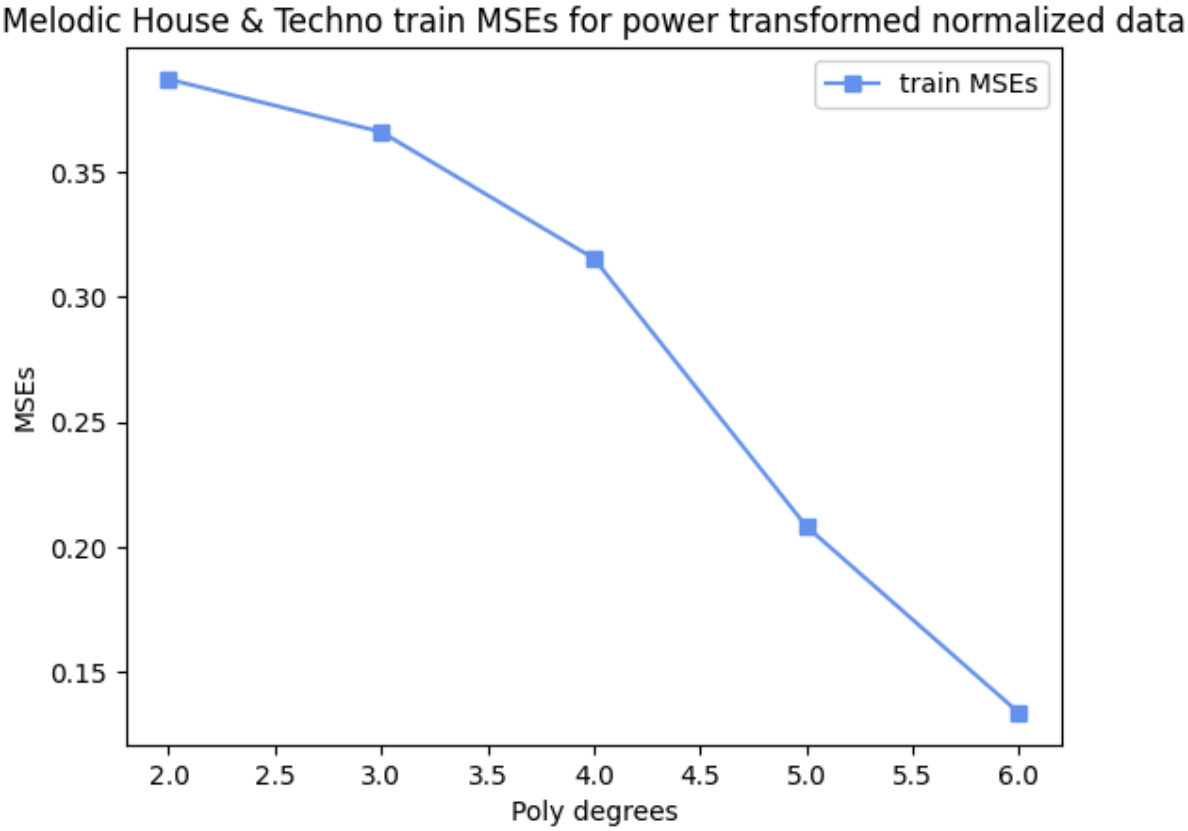
```
r2_5_techno_test = r2_score(power_techno_ytest, y_poly_pred5_test)
mse_5_techno_test = mean_squared_error(power_techno_ytest, y_poly_pred5_test
pearson_corr_5_techno_test, _ = pearsonr(power_techno_ytest, y_poly_pred5_te
plt.figtext(0.6, 0.28, f"R^2: {r2_5_techno_test:.4f}\nMSE: {mse_5_techno_tes

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

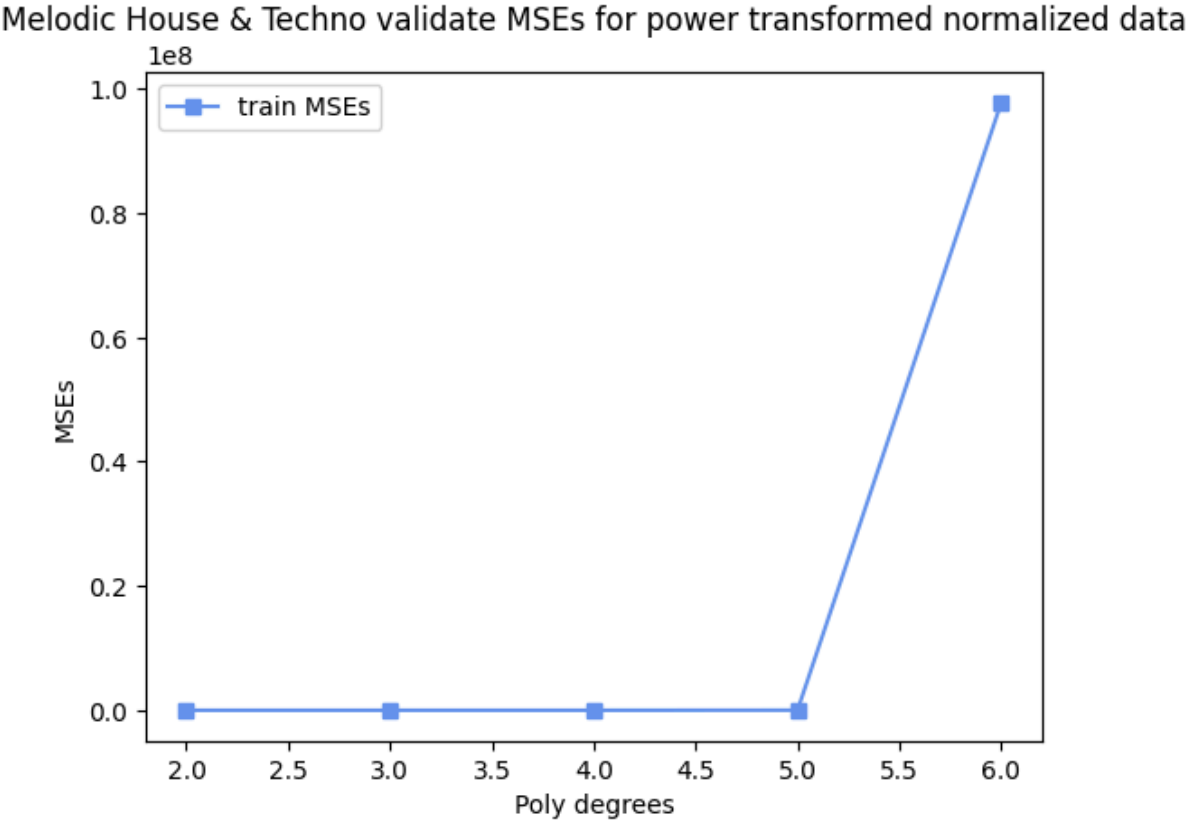


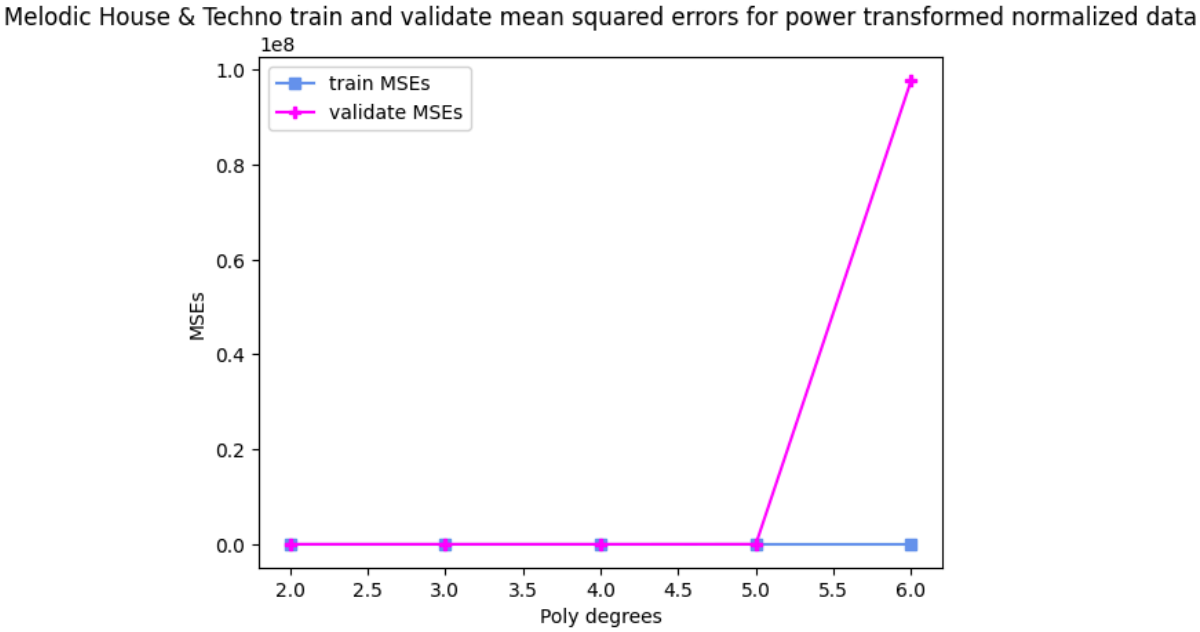
Melodic House and Techno

In [28]:
```
power_melodic_degrees = [2, 3, 4, 5, 6]
power_melodic_genre_mask = (power_scaling_train[:, 3] == 90.)
power_melodic_genre_validate_mask = (power_scaling_validate[:, 3] == 90.)
power_melodic_genre_test_mask = (power_scaling_test[:, 3] == 90.)


power_melodic_xtrain = power_scaling_train[power_melodic_genre_mask, 4:12]
power_melodic_ytrain = power_scaling_train[power_melodic_genre_mask, 12]

power_melodic_xvalidate = power_scaling_validate[power_melodic_genre_validat
power_melodic_yvalidate = power_scaling_validate[power_melodic_genre_validat

power_melodic_xtest = power_scaling_test[power_melodic_genre_test_mask, 4:12
power_melodic_ytest = power_scaling_test[power_melodic_genre_test_mask, 12]

power_melodic_train_mses, power_melodic_validate_mses = poly_degree_regress(

# Print results
print("melodic training MSEs:", power_melodic_train_mses)
print("melodic validation MSEs:", power_melodic_validate_mses)


melodic = subset_genres_dict[90]
melodic_title = f"{melodic} train MSEs for power transformed normalized data
validate_title =  f"{melodic} validate MSEs for power transformed normalized
melodic_overall_title = f"{melodic} train and validate mean squared errors f
plot_train_cv_mses(melodic_degrees, melodic_train_mses, melodic_title)
plot_train_cv_mses(melodic_degrees, melodic_validate_mses, validate_title)

train_validate_mses_2plots(melodic_degrees, melodic_train_mses, melodic_vali
```

```
melodic training MSEs: [0.45796257254081807, 0.4355855691636298, 0.373050224
1124543, 0.2246587379500196, 0.06757265057000462]
melodic validation MSEs: [0.4812746335052081, 0.5052346068477466, 0.80601737
20797963, 23.428400149179243, 4.553662519941645e+22]
```


Melodic House & Techno train MSEs for power transformed normalized data
train MSEs
MSEs
0.35
0.30
0.25
0.20
0.15
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Poly degrees

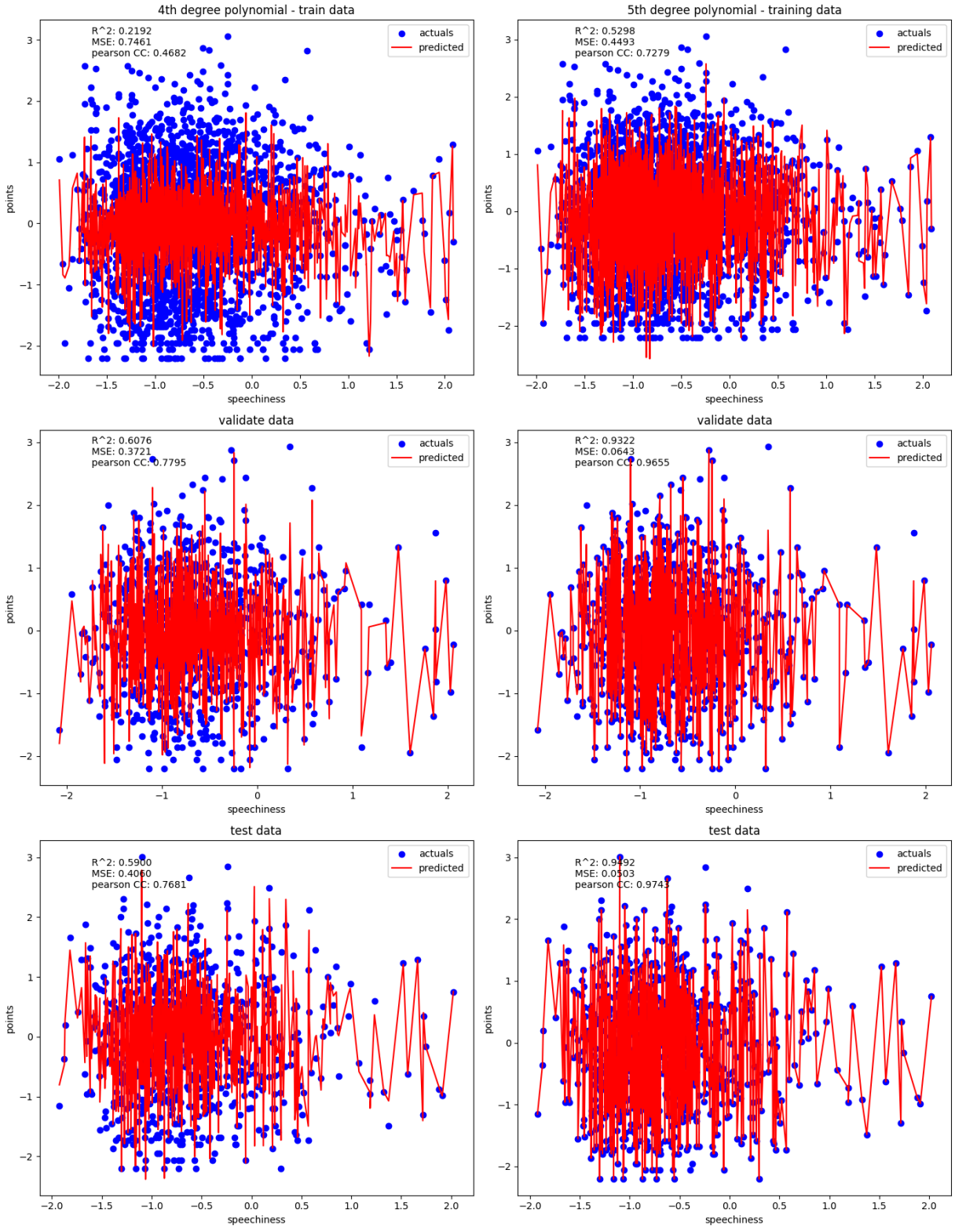

Melodic House & Techno validate MSEs for power transformed normalized data
1e8
train MSEs
MSEs
1.0
0.8
0.6
0.4
0.2
0.0
2.0
2.5
3.0
3.5
4.0
4.5
5.0
5.5
6.0
Poly degrees



In [29]:
```
power_melodic_poly4 = PolynomialFeatures(degree=4)
power_melodic_x_poly4_train = power_melodic_poly4.fit_transform(power_melodi
power_melodic_x_poly4_validate = power_melodic_poly4.fit_transform(power_mel
power_melodic_x_poly4_test = power_melodic_poly4.fit_transform(power_melodic

power_melodic_model4 = linear_model.LinearRegression()
power_melodic_model4.fit(power_melodic_x_poly4_train, power_melodic_ytrain)
y_poly_pred4_train = power_melodic_model4.predict(power_melodic_x_poly4_trai

power_melodic_model4.fit(power_melodic_x_poly4_validate, power_melodic_yvali
y_poly_pred4_validate = power_melodic_model4.predict(power_melodic_x_poly4_v

power_melodic_model4.fit(power_melodic_x_poly4_test, power_melodic_ytest)
y_poly_pred4_test = power_melodic_model4.predict(power_melodic_x_poly4_test)

power_melodic_sorted_indices_train = lumpnump.argsort(power_melodic_xtrain[:
power_melodic_sorted_indices_validate = lumpnump.argsort(power_melodic_xvali
power_melodic_sorted_indices_test = lumpnump.argsort(power_melodic_xtest[:,

power_melodic_sorted_xtrain_train = power_melodic_xtrain[power_melodic_sorte
power_melodic_sorted_xtrain_validate = power_melodic_xvalidate[power_melodic
power_melodic_sorted_xtrain_test = power_melodic_xtest[power_melodic_sorted_

power_melodic_sorted_ytrain_train = power_melodic_ytrain[power_melodic_sorte
power_melodic_sorted_ytrain_validate = power_melodic_yvalidate[power_melodic
power_melodic_sorted_ytrain_test = power_melodic_ytest[power_melodic_sorted_

power_melodic_sorted_y_poly_pred4_train = y_poly_pred4_train[power_melodic_s
power_melodic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_mel
power_melodic_sorted_y_poly_pred4_test = y_poly_pred4_test[power_melodic_sor

# Polynomial regression for degree 5
power_melodic_poly5 = PolynomialFeatures(degree=5)
power_melodic_x_poly5_train = power_melodic_poly5.fit_transform(power_melodi
power_melodic_x_poly5_validate = power_melodic_poly5.fit_transform(power_mel
power_melodic_x_poly5_test = power_melodic_poly5.fit_transform(power_melodic
```

```
# Fitting the polynomial regression model for degree 5
power_melodic_model5 = linear_model.LinearRegression()
power_melodic_model5.fit(power_melodic_x_poly5_train, power_melodic_ytrain)
y_poly_pred5_train = power_melodic_model5.predict(power_melodic_x_poly5_trai

power_melodic_model5.fit(power_melodic_x_poly5_validate, power_melodic_yvali
y_poly_pred5_validate = power_melodic_model5.predict(power_melodic_x_poly5_v

power_melodic_model5.fit(power_melodic_x_poly5_test, power_melodic_ytest)
y_poly_pred5_test = power_melodic_model5.predict(power_melodic_x_poly5_test)

power_melodic_sorted_y_poly_pred5_train = y_poly_pred5_train[power_melodic_s
power_melodic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_mel
power_melodic_sorted_y_poly_pred5_test = y_poly_pred5_test[power_melodic_sor

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_melodic_sorted_xtrain_train, power_melodic_sorted_ytrain_t
plt.plot(power_melodic_sorted_xtrain_train, power_melodic_sorted_y_poly_pred
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_melodic_train = r2_score(power_melodic_ytrain, y_poly_pred4_train)
mse_4_melodic_train = mean_squared_error(power_melodic_ytrain, y_poly_pred4_
pearson_corr_4_melodic_train, _ = pearsonr(power_melodic_ytrain, y_poly_pred
plt.figtext(0.1, 0.95, f"R^2: {r2_4_melodic_train:.4f}\nMSE: {mse_4_melodic_

plt.subplot(3, 2, 3)
plt.scatter(power_melodic_sorted_xtrain_validate, power_melodic_sorted_ytrai
plt.plot(power_melodic_sorted_xtrain_validate, power_melodic_sorted_y_poly_p
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_melodic_validate = r2_score(power_melodic_yvalidate, y_poly_pred4_valid
mse_4_melodic_validate = mean_squared_error(power_melodic_yvalidate, y_poly_
pearson_corr_4_melodic_validate, _ = pearsonr(power_melodic_yvalidate, y_pol
plt.figtext(0.1, 0.62, f"R^2: {r2_4_melodic_validate:.4f}\nMSE: {mse_4_meloc

plt.subplot(3, 2, 5)
plt.scatter(power_melodic_sorted_xtrain_test, power_melodic_sorted_ytrain_te
plt.plot(power_melodic_sorted_xtrain_test, power_melodic_sorted_y_poly_pred4
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_melodic_test = r2_score(power_melodic_ytest, y_poly_pred4_test)
mse_4_melodic_test = mean_squared_error(power_melodic_ytest, y_poly_pred4_te
pearson_corr_4_melodic_test, _ = pearsonr(power_melodic_ytest, y_poly_pred4_
plt.figtext(0.1, 0.28, f"R^2: {r2_4_melodic_test:.4f}\nMSE: {mse_4_melodic_t
```

```
# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_melodic_sorted_xtrain_train, power_melodic_sorted_ytrain_t
plt.plot(power_melodic_sorted_xtrain_train, power_melodic_sorted_y_poly_prec
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_melodic_train = r2_score(power_melodic_ytrain, y_poly_pred5_train)
mse_5_melodic_train = mean_squared_error(power_melodic_ytrain, y_poly_pred5_
pearson_corr_5_melodic_train, _ = pearsonr(power_melodic_ytrain, y_poly_prec
plt.figtext(0.6, 0.95, f"R^2: {r2_5_melodic_train:.4f}\nMSE: {mse_5_melodic_

plt.subplot(3, 2, 4)
plt.scatter(power_melodic_sorted_xtrain_validate, power_melodic_sorted_ytrai
plt.plot(power_melodic_sorted_xtrain_validate, power_melodic_sorted_y_poly_p
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_melodic_validate = r2_score(power_melodic_yvalidate, y_poly_pred5_valic
mse_5_melodic_validate = mean_squared_error(power_melodic_yvalidate, y_poly_
pearson_corr_5_melodic_validate, _ = pearsonr(power_melodic_yvalidate, y_pol
plt.figtext(0.6, 0.62, f"R^2: {r2_5_melodic_validate:.4f}\nMSE: {mse_5_meloc

plt.subplot(3, 2, 6)
plt.scatter(power_melodic_sorted_xtrain_test, power_melodic_sorted_ytrain_te
plt.plot(power_melodic_sorted_xtrain_test, power_melodic_sorted_y_poly_pred5
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_5_melodic_test = r2_score(power_melodic_ytest, y_poly_pred5_test)
mse_5_melodic_test = mean_squared_error(power_melodic_ytest, y_poly_pred5_te
pearson_corr_5_melodic_test, _ = pearsonr(power_melodic_ytest, y_poly_pred5_
plt.figtext(0.6, 0.28, f"R^2: {r2_5_melodic_test:.4f}\nMSE: {mse_5_melodic_t

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

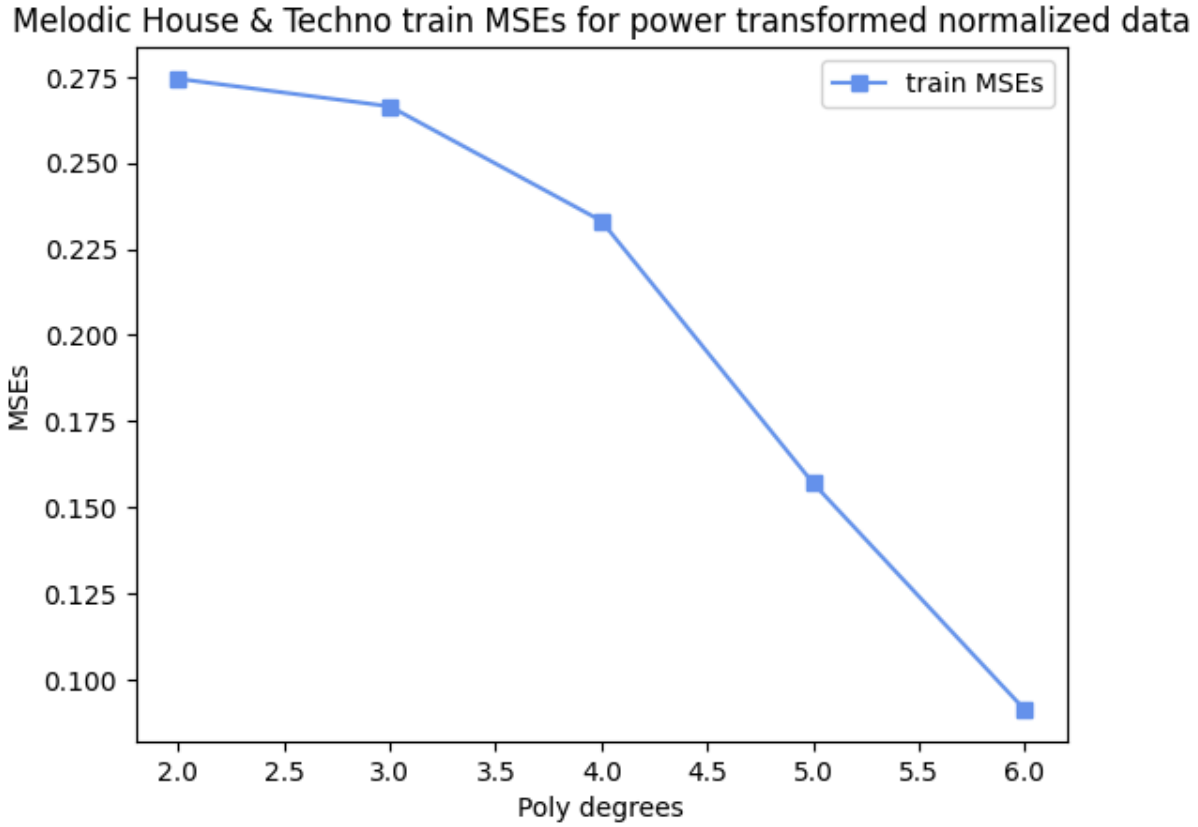


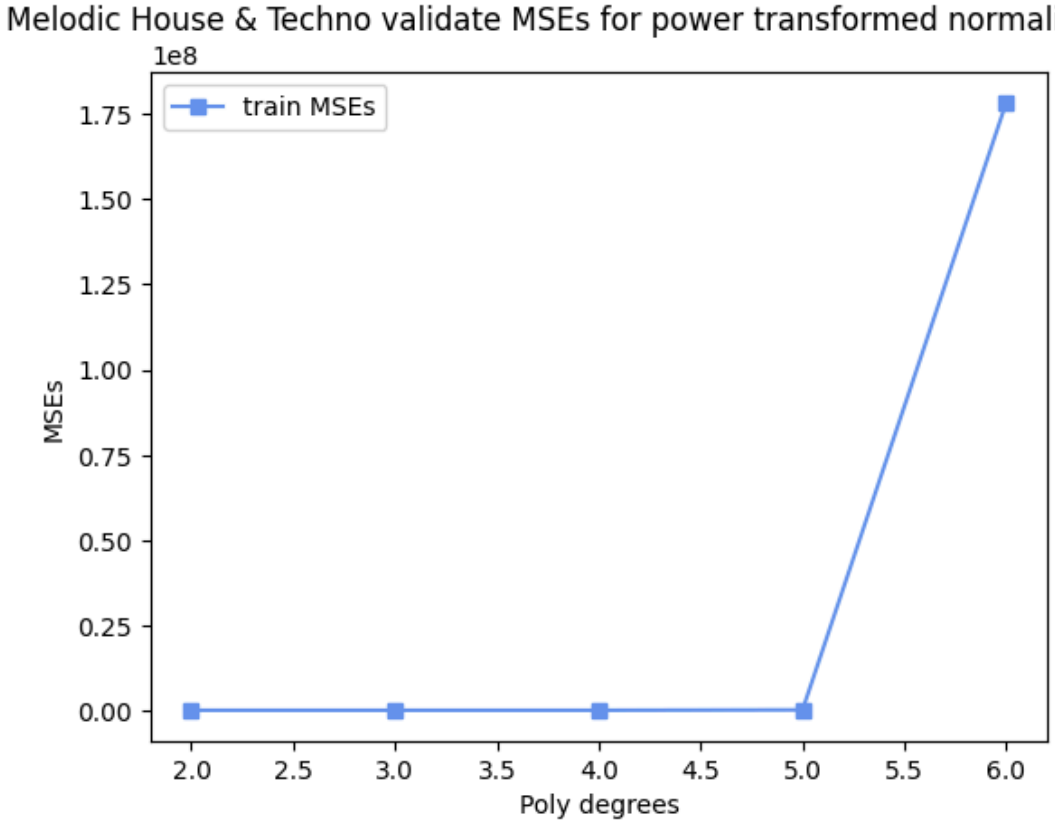
Organic house/down tempo

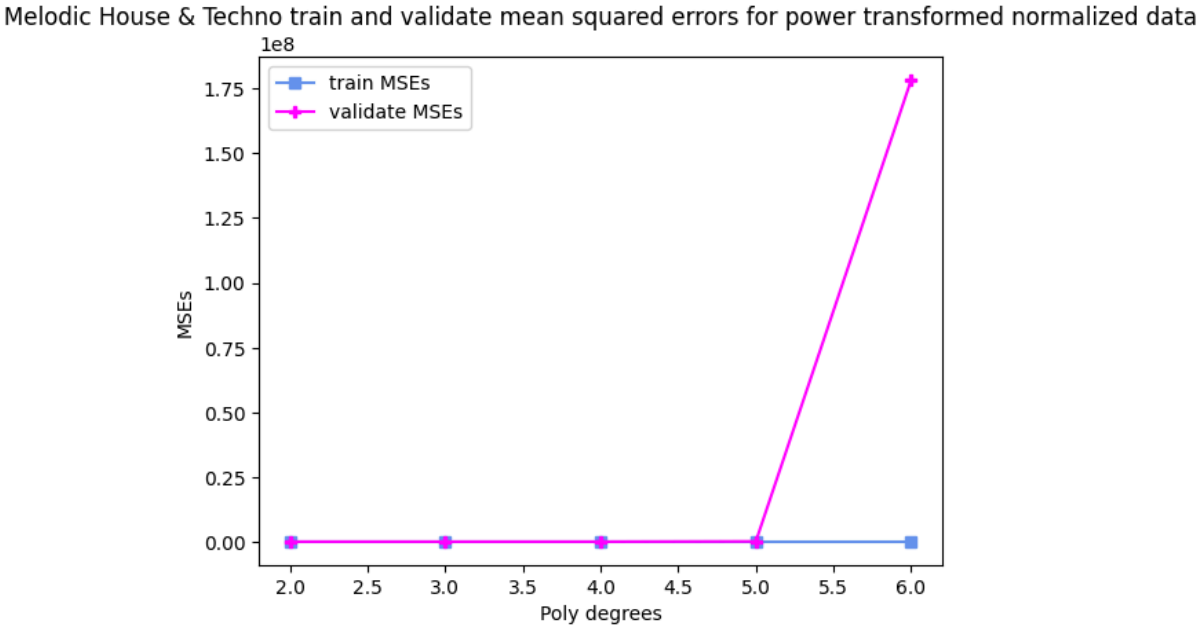
In [30]:
```
power_organic_degrees = [2, 3, 4, 5, 6]
power_organic_genre_mask = (power_scaling_train[:, 3] == 93.)
power_organic_genre_validate_mask = (power_scaling_validate[:, 3] == 93.)
power_organic_genre_test_mask = (power_scaling_test[:, 3] == 93.)
```

```
power_organic_xtrain = power_scaling_train[power_organic_genre_mask, 4:12]
power_organic_ytrain = power_scaling_train[power_organic_genre_mask, 12]

power_organic_xvalidate = power_scaling_validate[power_organic_genre_validat
power_organic_yvalidate = power_scaling_validate[power_organic_genre_validat

power_organic_xtest = power_scaling_test[power_organic_genre_test_mask, 4:12
power_organic_ytest = power_scaling_test[power_organic_genre_test_mask, 12]

power_organic_train_mses, power_organic_validate_mses = poly_degree_regress(

# Print results
print("organic training MSEs:", power_organic_train_mses)
print("organic validation MSEs:", power_organic_validate_mses)


organic = subset_genres_dict[90]
organic_title = f"{organic} train MSEs for power transformed normalized data
validate_title =  f"{organic} validate MSEs for power transformed normalized
organic_overall_title = f"{organic} train and validate mean squared errors f
plot_train_cv_mses(organic_degrees, organic_train_mses, organic_title)
plot_train_cv_mses(organic_degrees, organic_validate_mses, validate_title)

train_validate_mses_2plots(organic_degrees, organic_train_mses, organic_vali
```

```
organic training MSEs: [0.45877365350554633, 0.4326623689893219, 0.366429670
43637786, 0.21876859681057234, 0.05865944594822419]
organic validation MSEs: [0.5029782560866763, 0.5304873012992795, 0.97385942
46910489, 256.7192975346026, 12009.450594779257]
```





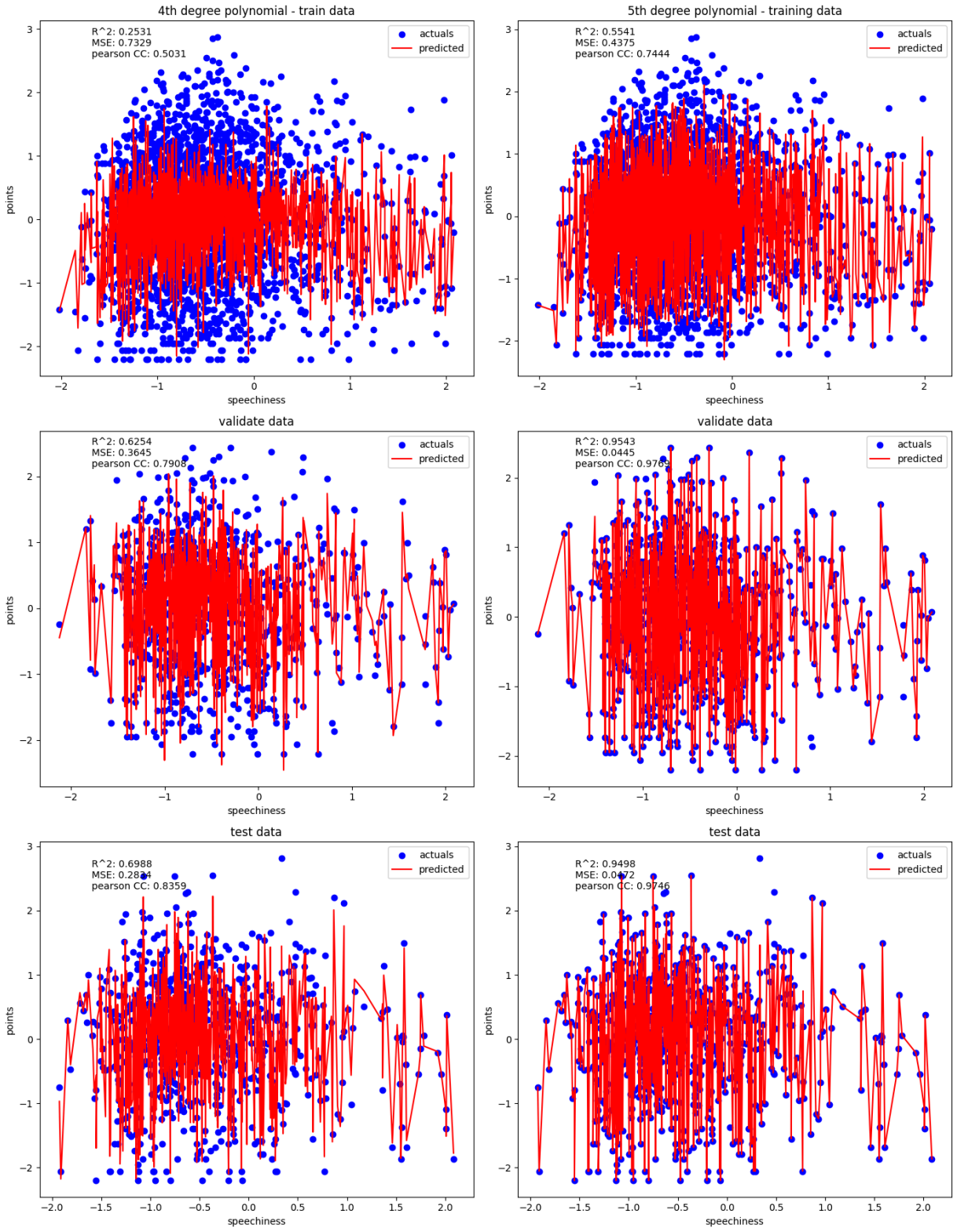


In [31]:
```
power_organic_poly4 = PolynomialFeatures(degree=4)
power_organic_x_poly4_train = power_organic_poly4.fit_transform(power_organi
power_organic_x_poly4_validate = power_organic_poly4.fit_transform(power_org
power_organic_x_poly4_test = power_organic_poly4.fit_transform(power_organic

power_organic_model4 = linear_model.LinearRegression()
power_organic_model4.fit(power_organic_x_poly4_train, power_organic_ytrain)
y_poly_pred4_train = power_organic_model4.predict(power_organic_x_poly4_trai

power_organic_model4.fit(power_organic_x_poly4_validate, power_organic_yvali
y_poly_pred4_validate = power_organic_model4.predict(power_organic_x_poly4_v
```

```
power_organic_model4.fit(power_organic_x_poly4_test, power_organic_ytest)
y_poly_pred4_test = power_organic_model4.predict(power_organic_x_poly4_test)

power_organic_sorted_indices_train = lumpnump.argsort(power_organic_xtrain[:
power_organic_sorted_indices_validate = lumpnump.argsort(power_organic_xvali
power_organic_sorted_indices_test = lumpnump.argsort(power_organic_xtest[:,

power_organic_sorted_xtrain_train = power_organic_xtrain[power_organic_sorte
power_organic_sorted_xtrain_validate = power_organic_xvalidate[power_organic
power_organic_sorted_xtrain_test = power_organic_xtest[power_organic_sorted_

power_organic_sorted_ytrain_train = power_organic_ytrain[power_organic_sorte
power_organic_sorted_ytrain_validate = power_organic_yvalidate[power_organic
power_organic_sorted_ytrain_test = power_organic_ytest[power_organic_sorted_

power_organic_sorted_y_poly_pred4_train = y_poly_pred4_train[power_organic_s
power_organic_sorted_y_poly_pred4_validate = y_poly_pred4_validate[power_org
power_organic_sorted_y_poly_pred4_test = y_poly_pred4_test[power_organic_sor

# Polynomial regression for degree 5
power_organic_poly5 = PolynomialFeatures(degree=5)
power_organic_x_poly5_train = power_organic_poly5.fit_transform(power_organi
power_organic_x_poly5_validate = power_organic_poly5.fit_transform(power_org
power_organic_x_poly5_test = power_organic_poly5.fit_transform(power_organic

# Fitting the polynomial regression model for degree 5
power_organic_model5 = linear_model.LinearRegression()
power_organic_model5.fit(power_organic_x_poly5_train, power_organic_ytrain)
y_poly_pred5_train = power_organic_model5.predict(power_organic_x_poly5_trai

power_organic_model5.fit(power_organic_x_poly5_validate, power_organic_yvali
y_poly_pred5_validate = power_organic_model5.predict(power_organic_x_poly5_v

power_organic_model5.fit(power_organic_x_poly5_test, power_organic_ytest)
y_poly_pred5_test = power_organic_model5.predict(power_organic_x_poly5_test)

power_organic_sorted_y_poly_pred5_train = y_poly_pred5_train[power_organic_s
power_organic_sorted_y_poly_pred5_validate = y_poly_pred5_validate[power_org
power_organic_sorted_y_poly_pred5_test = y_poly_pred5_test[power_organic_sor

# Plotting results
plt.figure(figsize=(14, 18))

# Degree 4 plots
plt.subplot(3, 2, 1)
plt.scatter(power_organic_sorted_xtrain_train, power_organic_sorted_ytrain_t
plt.plot(power_organic_sorted_xtrain_train, power_organic_sorted_y_poly_pred
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('4th degree polynomial - train data')
r2_4_organic_train = r2_score(power_organic_ytrain, y_poly_pred4_train)
mse_4_organic_train = mean_squared_error(power_organic_ytrain, y_poly_pred4_
pearson_corr_4_organic_train, _ = pearsonr(power_organic_ytrain, y_poly_pred
plt.figtext(0.1, 0.95, f"R^2: {r2_4_organic_train:.4f}\nMSE: {mse_4_organic_
```

```
plt.subplot(3, 2, 3)
plt.scatter(power_organic_sorted_xtrain_validate, power_organic_sorted_ytrai
plt.plot(power_organic_sorted_xtrain_validate, power_organic_sorted_y_poly_p
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_4_organic_validate = r2_score(power_organic_yvalidate, y_poly_pred4_valic
mse_4_organic_validate = mean_squared_error(power_organic_yvalidate, y_poly_
pearson_corr_4_organic_validate, _ = pearsonr(power_organic_yvalidate, y_pol
plt.figtext(0.1, 0.62, f"R^2: {r2_4_organic_validate:.4f}\nMSE: {mse_4_organ

plt.subplot(3, 2, 5)
plt.scatter(power_organic_sorted_xtrain_test, power_organic_sorted_ytrain_te
plt.plot(power_organic_sorted_xtrain_test, power_organic_sorted_y_poly_pred4
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('test data')
r2_4_organic_test = r2_score(power_organic_ytest, y_poly_pred4_test)
mse_4_organic_test = mean_squared_error(power_organic_ytest, y_poly_pred4_te
pearson_corr_4_organic_test, _ = pearsonr(power_organic_ytest, y_poly_pred4_
plt.figtext(0.1, 0.28, f"R^2: {r2_4_organic_test:.4f}\nMSE: {mse_4_organic_t

# Degree 5 plots
plt.subplot(3, 2, 2)
plt.scatter(power_organic_sorted_xtrain_train, power_organic_sorted_ytrain_t
plt.plot(power_organic_sorted_xtrain_train, power_organic_sorted_y_poly_pred
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('5th degree polynomial - training data')
r2_5_organic_train = r2_score(power_organic_ytrain, y_poly_pred5_train)
mse_5_organic_train = mean_squared_error(power_organic_ytrain, y_poly_pred5_
pearson_corr_5_organic_train, _ = pearsonr(power_organic_ytrain, y_poly_pred
plt.figtext(0.6, 0.95, f"R^2: {r2_5_organic_train:.4f}\nMSE: {mse_5_organic_

plt.subplot(3, 2, 4)
plt.scatter(power_organic_sorted_xtrain_validate, power_organic_sorted_ytrai
plt.plot(power_organic_sorted_xtrain_validate, power_organic_sorted_y_poly_p
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
plt.title('validate data')
r2_5_organic_validate = r2_score(power_organic_yvalidate, y_poly_pred5_valic
mse_5_organic_validate = mean_squared_error(power_organic_yvalidate, y_poly_
pearson_corr_5_organic_validate, _ = pearsonr(power_organic_yvalidate, y_pol
plt.figtext(0.6, 0.62, f"R^2: {r2_5_organic_validate:.4f}\nMSE: {mse_5_organ

plt.subplot(3, 2, 6)
plt.scatter(power_organic_sorted_xtrain_test, power_organic_sorted_ytrain_te
plt.plot(power_organic_sorted_xtrain_test, power_organic_sorted_y_poly_pred5
plt.xlabel('speechiness')
plt.ylabel('points')
plt.legend()
```

```
plt.title('test data')
r2_5_organic_test = r2_score(power_organic_ytest, y_poly_pred5_test)
mse_5_organic_test = mean_squared_error(power_organic_ytest, y_poly_pred5_te
pearson_corr_5_organic_test, _ = pearsonr(power_organic_ytest, y_poly_pred5_
plt.figtext(0.6, 0.28, f"R^2: {r2_5_organic_test:.4f}\nMSE: {mse_5_organic_t

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



In [ ]:
```
#by genre
for genre_id in subset_genres:
    genre_name = subset_genres_dict.get(genre_id, f"Genre {genre_id}")
    toptracks_bpmeta_matrix = log_train[log_train[:, toptracks_bpmeta_matrix
    x_features = toptracks_bpmeta_matrix[:, core_x_features]
    for i, feature_index in enumerate(core_x_features):
        feature = toptracks_bpmeta_matrix[:, feature_index]
        feature_name = list(toptracks_bpmeta_matrix_df_dict.keys())[list(top
        title = f"{feature_name} for {genre_name}"
        log_plot(feature, title)
```

In [ ]:
```
#by label
for label_id in subset_labels:
    label_name = label_ids_dict.get(label_id, f"Label {label_id}")
    label_matrix = log_train[log_train[:, toptracks_bpmeta_matrix_df_dict['l
    x_features = label_matrix[:, core_x_features]
    for i, feature_index in enumerate(core_x_features):
        feature = label_matrix[:, feature_index]
        feature_name = list(toptracks_bpmeta_matrix_df_dict.keys())[list(top
        title = f"{feature_name} for {label_name}"
        log_plot(feature, title)
```

In [ ]:
```
run_all_calculations_label(zscore_train, zscore_validate, zscore_test)
```

In [ ]:
```
run_all_calculations_genre(mean_train, mean_validate, mean_test)
```